Andreas Germann

Evaluation of AQM Schemes to Support Low Latency in the Internet

18. September 2017

Master Thesis MA-2017-07
July 2017 to September 2017

Tutor: Prof. Dr. Laurent Vanbever
Supervisors: Mirja Kuehlewind, Brian Trammell
Acknowledgements

I want to thank my tutor Prof. Dr. Laurent Vanbever and my supervisors Mirja Kuehlewind and Brian Trammel for their support while working on this project.
Abstract

In this work, we evaluated different AQM schemes that can be used in the internet. Since low latency becomes a requirement for increasing amounts of applications and traffic and since queuing remains a significant source of latency, we wanted to find out how well these AQM schemes would perform with respect to low latency requirements.

For these evaluations, we implemented different AQM schemes for a network simulator and ran simulations. Based on these results we evaluated the performance of these schemes with respect to queuing delays and link utilization.

Our results showed that simple AQM with a single queue only rarely could achieve low queuing delays and that they had to sacrifice link utilization when trying. Some more complex schemes with two queues also had a hard time achieving low queuing delays but did not loose link utilization when trying. Other schemes with two queues were able to achieve very low queuing delays but also had to sacrifice link utilization.
## Contents

1 Introduction .................................................. 7

2 Background .................................................. 9
   2.1 Explicit Congestion Notification .............................. 9
   2.2 TCP Congestion Control ...................................... 9
      2.2.1 Classic Congestion Control ................................. 10
      2.2.2 Scalable Congestion Control ............................... 12
      2.2.3 Scalable vs Classic ..................................... 13
   2.3 Active Queue Management .................................... 14
      2.3.1 Random Early Detection ................................. 15
      2.3.2 Curvy Random Early Detection .............................. 15
      2.3.3 Proportional Integral Controller Enhanced ............... 16
      2.3.4 Proportional Integral Controller Squared ................. 17
   2.4 Low Latency Low Loss Scalable Throughput ................. 18
      2.4.1 Basic Concept ........................................ 18
      2.4.2 DualQ Coupled AQM .................................... 18
   2.5 Network Simulator 3 ....................................... 21
      2.5.1 Direct Code Execution .................................. 21
      2.5.2 Link Models ............................................. 21
   2.6 Jain's Fairness Index .................................... 22

3 Simulation .................................................. 23
   3.1 AQM Implementations ....................................... 23
      3.1.1 Single Queue AQM ...................................... 23
      3.1.2 Dual Queue Uncoupled AQM ................................. 24
      3.1.3 DualQ Coupled AQM .................................... 25
   3.2 Kernel Modifications ....................................... 27
   3.3 Simulation Network ....................................... 27
      3.3.1 Static Link Model ...................................... 27
      3.3.2 LTE Network ............................................. 28
   3.4 Simulation Scenarios ....................................... 28

4 Results .................................................. 31
   4.1 Metrics ................................................ 31
   4.2 Static Link Simulations .................................... 31
      4.2.1 Single Queue AQM ...................................... 32
      4.2.2 Dual Queue Uncoupled AQM ................................. 34
      4.2.3 DualQ Coupled AQM .................................... 38
      4.2.4 Comparison ............................................. 43
   4.3 LTE Link Simulations ....................................... 44
      4.3.1 Single Queue AQM ...................................... 44
      4.3.2 Dual Queue Uncoupled AQM ................................. 46
      4.3.3 DualQ Coupled AQM .................................... 49
      4.3.4 Comparison ............................................. 50
   4.4 Conclusion ............................................. 52
      4.4.1 AQM Schemes ........................................ 52
      4.4.2 TCP Settings ........................................... 52
Chapter 1

Introduction

In today's internet, delays are increasingly becoming a problem. More and more applications and traffic require low latencies to provide a satisfying service. Examples for such applications are voice over IP, video telephony or interactive web applications. Recent efforts to reduce latency by placing servers closer to the users were of limited success since queuing persists as an intermittent source of delay. Queuing causes spikes in latency when buffers fill up due to congestion. Even when using modern active queue management (AQM), queueing can still cause significant delays. [26]

AQM are algorithms that control the size and thereby also the delay of a queue through pre-emptive drops. This leads to a trade-off of latency against throughput. Low queuing delays are achieved by reducing the amount of packets in the queues. This can leave the queue temporarily empty which leads to underutilization of the link and loss of throughput. Allowing high queuing delays results in higher amounts of packets in the queues and higher link utilization.

Low Latency Low Loss Scalable Throughput (L4S) is a proposed algorithm that provides an ultra low latency service in parallel to the classic best-effort service. This is done by separating incoming traffic into the two classes based on an identifier and then applying different control schemes to achieve the respective goal. To achieve ultra low latency the low-latency class is given priority over the best-effort class. [11]

In this project we implement different AQM schemes in a network simulator. With these implementations, we run simulations in different scenarios to gather data on the performance of these schemes. Based on the results of these simulations we will evaluate the performance of the different algorithms. This should give insight into their advantages, drawbacks and problems.
Chapter 2

Background

2.1 Explicit Congestion Notification

Explicit Congestion Notification (ECN) is an extension of TCP/IP [15]. ECN allows a router to send congestion signals without dropping packets. To do that, a router marks packets instead of dropping them.

If a router wants to send a congestion signal, it marks a packet. Upon receiving a marked packet, the endpoint of a connection informs the sender by marking the acknowledgement packet.

Using ECN implicates a problem with fairness called ECN unfairness. ECN unfairness describes different scenarios where bandwidth is distributed unfairly between flows because of ECN. Corrupted ECN signals and the ignoring of ECN feedback are mentioned as reasons for ECN unfairness. But there is a case without signal corruption or disobedience to feedback that can cause ECN unfairness. In this case, non-ECN flows receive less bandwidth than ECN flows because their packets are dropped instead of marked. This happens especially often in situations where an AQM reaches high drop probabilities.

The RED algorithm described in Section 2.3.1 is mentioned as an example. When the queue length exceeds the maximum threshold, only ECN flow receive throughput since all non-ECN packets are dropped.

2.2 TCP Congestion Control

TCP congestion controls are algorithms designed to control the sending rate of network devices [8]. Their goal is to adjust the sending rate according to the currently available bandwidth. On one hand, they should keep the sending rate high in order to achieve a high link utilization. On the other hand, they should keep the rate low enough to avoid congestions and congestive breakdowns.

The sending rate in TCP is controlled by the congestion window. The congestion window defines how many packets a connection is allowed to have in flight simultaneously. If the congestion window is full, the sender has to wait for acknowledgements before he can send further packets. Congestion controls define how the congestion window is changed. The difference between congestion controls is how and when they change the congestion window. Generally, the absence of packet losses indicates that the network is not congested and the congestion window can be increased. When a packet loss is detected, this indicates a congestion and the congestion window has to be reduced. Like losses, ECN feedback is also a congestion signal and should lead to a congestion window reduction.

A lot of congestion controls use slow-start, an algorithm for increasing the congestion window. It is comprised of two phases, the slow-start phase and the congestion-avoidance phase. In the slow-start phase, the congestion window is increased quickly in order to bring the connection
up to speed. In the congestion avoidance phase, the window is increased slowly. Also, a lot of algorithms use fast-recovery/fast-retransmission, an algorithm that can sometimes skip the slow-start phase after a loss. If the sender registers a packet the loss of a specific packet, it can perform a fast retransmission of this packet. If it succeeds, the slow-start phase is skipped, the congestion window is increased and the control continues in the congestion-avoidance phase.

Since this work is about congestion and since the congestions controls in our simulations will mostly be in the congestion-avoidance phase, we will focus on the congestion-avoidance behaviour of the algorithms.

### 2.2.1 Classic Congestion Control

One class of congestion controls that we used in our simulations are classic congestion controls [16] [24]. Classic congestion controls are in the sense classic that they the ones currently being used in today’s internet. They work by the AIMD (additive increase/multiplicative decrease) principle. The main difference to scalable congestion controls explained in Section 2.2.2 is that the congestion window reduction is quite aggressive when congestion signals are registered.

**Reno**

Reno is a simple classic congestion control algorithm [8]. Reno uses slow-start and fast-recovery/fast-retransmission.

In the congestion-avoidance phase and the absence of losses, it increases the congestion window by one over the current congestion window for every acknowledgement received. This leads to an effective increase of one packet every round-trip-time.

In case of losses or the receiving of ECN feedback, the congestion window is reduced. Simplified, it is halved. Actually, it is set set to one packet but due fast-recovery/fast-retransmission, the slow-start phase can be skipped and the congestion window restored to half its previous value, continuing in the congestion-avoidance phase.

Figure 2.1 shows an example plot of the Reno congestion window over time in steady state.

**Cubic**

Cubic is another classic congestion control [17] [24]. It also uses slow-start and fast-recovery/fast-retransmission.
2.2 TCP Congestion Control

Figure 2.2: Congestion window over time in Cubic

Like Reno, Cubic increases its congestion windows for every received acknowledgement. In the congestion-avoidance phase, Cubic uses a cubic congestion window growth function depending on the time since the last loss or ECN feedback. This function is shown in Equation 2.1. The factor $\beta$ is the decrease factor and is recommended to be 0.7. The factor $C$ defines the aggressiveness of the algorithm and is recommended to be 0.4. $W_{\text{max}}$ is the stored congestion window from before the last reduction.

$$W(t) = C \cdot (t - K)^3 + W_{\text{max}}$$

$$K = \sqrt[3]{\frac{W_{\text{max}} \cdot (1 - \beta)}{C}}$$

The cubic growth function has three regions. One region is the TCP-friendly region. Here, the growth function is compared to a standard TCP growth function (Equation 2.2) with $\beta_{\text{aimd}}$ equals 0.5. If $W(t)$ is smaller than $W_{\text{aimd}}(t)$, then the congestion window should be set to $W_{\text{aimd}}(t)$ whenever an acknowledgement is received. This should ensure that the congestion window in Cubic grows at least as fast as in the linearly growing standard TCP.

The other two regions are the convex and the concave region. In these two regions, the congestion window is set to $W(t + RTT)$ from Equation 2.1 for every acknowledgement received. As long as the congestion window has not reached $w_{\text{max}}$, the cubic function is in its concave region. Here it grows quickly at first and slows down when approaching $W_{\text{max}}$. Once the congestion window goes past $W_{\text{max}}$, it enters the convex region and grows increasingly fast.

$$W_{\text{aimd}}(t) = W_{\text{max}} \cdot \beta_{\text{aimd}} + 3 \cdot \frac{1 - \beta_{\text{aimd}}}{1 + \beta_{\text{aimd}}} \cdot \frac{t}{RTT}$$

The window reduction in Cubic works like the one in Reno with fast-recovery/fast-retransmission and slow-start. But simplified, the congestion window is reduced by 70%. Also, the congestion window before the reduction is saved as the new $W_{\text{max}}$. This is shown in Equation 2.3.

$$W_{\text{max}} = \text{cwnd}$$

$$\text{cwnd} = \text{cwnd} \cdot (1 - \text{beta})$$

Figure 2.2 shows an example plot of the Cubic congestion window over time in steady state.
2.2.2 Scalable Congestion Control

Another class of congestion controls are scalable congestion controls [5] [20]. Compared to classic congestion controls, scalable congestion controls reduce the congestion window more conservatively.

Data Center TCP

Data Center TCP (DCTCP) is a scalable congestion control [5] [7]. Like Reno, it also uses slow-start and fast-recovery/fast-retransmission.

In the congestion-avoidance phase, the congestion window is also increased by one packet every round-trip-time.

The difference lies in the congestion window decrease mechanism. DCTCP distinguishes between losses and ECN feedback. For ECN feedback, it reduces the congestion window based on an exponentially weighted moving average of the ratio between bytes where the acknowledgement was marked and bytes where the acknowledgement was not marked. For losses, DCTCP behaves like Reno, halving the congestion window.

The exponentially weighted moving average $\alpha$ is updated according to Equation 2.4 approximately once very round-trip-time. The factor $g$ sets the weight between old and the values. $g$ is recommended to be $\frac{1}{16}$.

Upon receiving an ECN feedback, DCTCP reduces the congestion window according to Equation 2.5. The reduction by the factor of $\frac{\alpha}{2}$ ensures that for high congestion when $\alpha$ is close to 1, DCTCP halves its congestion window like in the Reno algorithm.

\[
\alpha = (1 - g) \times \text{alpha} + g \times \frac{\text{marked_bytes_acknowledged}}{\text{bytes_acknowledged}}
\]

\[
cwnd = cwnd \times \left(1 - \frac{\alpha}{2}\right)
\]

Figure 2.3 shows an example plot of the DCTCP congestion window over time in steady state.

Relentless

Relentless is a simple implementation of a scalable congestion control [20] [21]. It also uses slow-start and fast-recovery/fast-retransmission.
2.2 TCP Congestion Control

The increase mechanism is again the same as in Reno, increasing the congestion window by one packet every round-trip-time.

The reduction mechanism of Relentless is very simple. The congestion window is reduced by one packet for every loss or ECN feedback. This is similar to DCTCP but leaves out the smoothing of the moving average function.

Figure 2.4 shows an example plot of the Relentless congestion window over time in steady state.

2.2.3 Scalable vs Classic

Buffer Size

Compared to classic congestion control, scalable congestion control offers three improvements. [7]. The first improvement is that scalable congestion control can achieve full link utilization with less buffer size than classic congestion control. The second, following from the first, is that scalable congestion control can achieve lower queuing delays since the buffer size can be smaller. And third, if a buffer is managed by an AQM, the lower buffer utilization of scalable congestion control leaves more capacity for bursts.

Figures 2.5a and 2.5b show example plots of the congestion windows of Reno and DCTCP in steady state with similar throughputs and round-trip-times. The required buffer sizes to achieve full link utilization are also plotted. Reno and DCTCP are used as examples for classic and scalable congestion controls. The same principle goes for other congestion controls in the respective classes.

Throughput Equation

One big problem with scalable congestion control is that it is not AIMD-friendly meaning that it starves out competing flows that use AIMD or classic congestion controls. This is due to the difference in throughput equations. [13] [14]

The throughput equations states the throughput of a flow that uses a certain congestion control. Different congestion controls have different throughput equations. The throughput equations for Reno, Cubic, DCTCP [13] and Relentless [14] are stated in Equations 2.6. They depend on the maximum segment size (MSS), the round-trip-time RTT and the drop rate $p$. 

![Figure 2.4: Congestion window over time in Relentless](image)
CHAPTER 2. BACKGROUND

(a) Congestion window and required buffer size for full link utilization in Reno
(b) Congestion window and required buffer size for full link utilization in DCTCP

Figure 2.5: Comparison of the congestion windows of Reno and Cubic

\[
\text{throughput}_{\text{reno}} = \frac{\text{MSS} \times 1.22}{\text{RTT} \times \text{drop\_rate}^{0.5}} \\
\text{throughput}_{\text{cubic}} = \frac{\text{MSS} \times 1.17}{\text{RTT}^{0.025} \times \text{drop\_rate}^{0.75}} \sim \frac{\text{MSS} \times 1.68}{\text{RTT} \times \text{drop\_rate}^{0.5}} \quad \text{(in TCP-friendly region)} \\
\text{throughput}_{\text{dctcp}} = \frac{\text{MSS} \times 2}{\text{RTT} \times \text{drop\_rate}} \\
\text{throughput}_{\text{relentless}} = \frac{\text{MSS} \times 0.49}{\text{RTT} \times \text{drop\_rate}} \tag{2.6}
\]

There is a major difference between the throughput equations of classic and scalable congestion controls. While the classic ones contain the drop rate as a square root, the scalable ones do contain it directly.

Assuming similar RTT and MSS and drop probability, the classic congestion controls have a much lower throughput than the scalable ones. That is the reason why flow with scalable congestion control starve out competing flows with classic congestion control if no special measures are adopted.

2.3 Active Queue Management

Active Queue Management (AQM) describes schemes that aim to control filling levels and delays of queues. [6] Queues in network devices serve as buffers to prevent losses when more packets come in than can go out. An ongoing congestion causes such queues to fill up. This can lead to massive drops in case of an overflow as well as large queuing delays.

AQM was originally developed to conserve throughput by preventing an excessive build-up of packets in case of a congestion. Such an excessive build-up can ultimately lead to the buffer overflowing resulting in massive packet loss. AQM controls the queue length by randomly dropping packets if it detects a congestion. This signals the senders to reduce their sending rate and mitigates the congestion. The cost of these pre-emptive drops should outweigh the cost of a buffer overflow with massive packet losses.

Later, AQM was also used to control queuing delays since latency became increasingly important and since queuing remained a significant source of delay. In this case, the goal is to keep the queue short in order to reduce delay. At the same time, the queue should not run empty because that would lead underutilization of the link. Controlling the queue length is a trade-off between latency and throughput.
2.3 Active Queue Management

The difference between AQM schemes is how the drop probability is implemented which determines how the queue length or delay is controlled.

If available, AQM can also use ECN marks instead of drops or a mixture of drops and marks.

2.3.1 Random Early Detection

Random Early Detection (RED) is a simple AQM [19] [12]. It randomly marks or drops packets with a drop probability depending on the current queue length.

RED uses three parameters. The minimum threshold defines the queue length up to which no packets are dropped or marked. The maximum threshold defines the queue length above which all packets are dropped or marked. The maximum drop probability defines the drop probability when the queue length is equal to the maximum threshold.

Between minimum and maximum threshold, the drop probability is a linear function that equals 0 at the minimum threshold and equals the maximum drop probability at the maximum threshold.

The maximum drop probability is recommended to be one or two percent. For the thresholds, there are no special recommendations.

Figure 2.6 shows the basic graph of the drop probability function of RED.

2.3.2 Curvy Random Early Detection

Curvy Random Early Detection (CRED) is another simple AQM [10]. It randomly drops or marks packets with a drop probability depending on the current queue delay. The AQM is called curvy because it uses an exponential function for the drop probability.

CRED uses two parameters to calculate the drop probability. The slope factor $D_q$ defines the queue delay above which all packets are dropped or marked. The curviness factor $U$ defines the exponent for the drop probability function.

The drop probability function is described in Equation 2.7. The drop probability is bounded to 1 for queue delays above the slope factor.

\[
drop\text{-}probability = \left(\frac{\text{queue\_delay}}{D_q}\right)^U
\]  

(2.7)

The current queue delay can either be measured by using timestamps or estimated using the current queue length and a measurement of the dequeuing rate. There are no recommenda-
2.3.3 Proportional Integral Controller Enhanced

Proportional Integral Controller Enhanced (PIE) is a more complex AQM [23] [22]. It uses current and past queuing delays for the calculation of the drop probability. The PIE algorithm consist of two parts.

The first part is the drop probability calculation. This function periodically updates the drop probability depending on the current, previous and target queue delay as well as the current drop probability. After the calculation, the current queuing delay is stored for the next calculation. Algorithm 1 shows the drop probability calculation as it is defined in [22].

The second part is the early drop function that decides whether a packet is dropped or marked. A packet is dropped or marked randomly using the calculated drop probability. Random dropping is prevented if the previous queuing delay and the drop probability are low enough or if the queue is almost empty. Random dropping is also prevented if PIE is in burst protection. Burst protection keeps PIE from dropping packets during short bursts. If the link is not congested for a while, burst protection is activated. In the case of a congestion, the burst protection deactivates after a certain time.

The current queue delay can either be measured by using timestamps or estimated using the current queue length and a measurement of the dequeuing rate.

These are recommended values for the PIE parameters from [22].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target delay</td>
<td>15 milliseconds</td>
</tr>
<tr>
<td>Burst protection period</td>
<td>150 milliseconds</td>
</tr>
<tr>
<td>Update period</td>
<td>15 milliseconds</td>
</tr>
<tr>
<td>Alpha</td>
<td>0.125</td>
</tr>
<tr>
<td>Beta</td>
<td>1.25</td>
</tr>
</tbody>
</table>

The full proposed pseudo code of PIE can be seen in [22].
2.3 Active Queue Management

\[ p = \alpha \times (\text{current\_delay} - \text{target\_delay}) + \beta \times (\text{current\_delay} - \text{previous\_delay}); \]

\[
\text{if } \text{drop\_prob} < 0.000001 \text{ then}
\]
\[
\quad p /= 2048;
\]
\[
\text{else if } \text{drop\_prob} < 0.00001 \text{ then}
\]
\[
\quad p /= 512;
\]
\[
\text{else if } \text{drop\_prob} < 0.001 \text{ then}
\]
\[
\quad p /= 128;
\]
\[
\text{else if } \text{drop\_prob} < 0.01 \text{ then}
\]
\[
\quad p /= 32;
\]
\[
\text{else if } \text{drop\_prob} < 0.1 \text{ then}
\]
\[
\quad p /= 8;
\]
\[
\text{else}
\]
\[
\quad \text{else}
\]
\[
\quad p = p;
\]
\[
\text{end}
\]
\[
\text{drop\_prob += p;}
\]
\[
\text{if } \text{cur\_queue\_delay} == 0 \text{ AND old\_queue\_delay} == 0 \text{ then}
\]
\[
\quad \text{drop\_prob} *= 0.98;
\]
\[
\text{end}
\]
\[
\text{if } \text{drop\_prob} < 0 \text{ then}
\]
\[
\quad \text{drop\_prob} = 0;
\]
\[
\text{end}
\]
\[
\text{previous\_delay}=\text{current\_delay};
\]

\text{Algorithm 1: Drop probability calculation of PIE from [22]}

2.3.4 Proportional Integral Controller Squared

Proportional Integral Controller Squared (PI2) is another AQM [13]. It is loosely based on PIE. A special thing about this AQM is that it can control scalable and classic congestion controls at the same time. This is done by adjusting the drop probability according to the congestion control. However, this required a reliable distinction between the congestion controls. The ECT(1) code-point is proposed for such a distinction.

The drop probability calculation is shown in Equation 2.8. The drop probability is updated periodically. The current queue delay can either be measured by using timestamps or estimated using the current queue length and a measurement of the dequeuing rate.

\[
drop\_prob = drop\_prob
+ \alpha \times \text{update\_period} \times (\text{cur\_queue\_delay} - \text{ref\_queue\_delay}) +
+ \beta \times \text{update\_period} \times (\text{cur\_queue\_delay} - \text{old\_queue\_delay}) \tag{2.8}
\]

After the distinction, the drop probability is adjusted for the congestion control. For a flow with classic congestion control, the drop probability is first scaled and then squared as shown in Equation 2.9. For scalable congestion controls, the drop probability would be left as it is.

\[
drop\_prob\_classic = \left(\frac{\text{drop\_prob}}{K}\right)^2 \tag{2.9}
\]

This different treatment accounts for the difference in the throughput equation of classic and scalable congestion controls described in Section 2.2.3 and allows them to coexist. The squaring adjusts the different powers of the drop rate in the throughput equations. The scaling shifts the throughput distribution to adjust for the constants. Two is recommended as scaling factor since it is the power of two that should achieve the fairest throughput distribution. Powers of two are favourable since divisions by two can be
implemented cheaply using bit shifts.

These are recommended values for the PI2 parameters from [22].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target delay</td>
<td>15 or 20 ms</td>
</tr>
<tr>
<td>Update period</td>
<td>16 or 32 ms</td>
</tr>
<tr>
<td>Alpha</td>
<td>10</td>
</tr>
<tr>
<td>Beta</td>
<td>100</td>
</tr>
<tr>
<td>K</td>
<td>2</td>
</tr>
</tbody>
</table>

2.4 Low Latency Low Loss Scalable Throughput

Low Latency Low Loss Scalable Throughput (L4S) is a proposed algorithm that aims to provide a low-latency service parallel to the current best-effort service [11]. In connection with L4S, we will call the low-latency service L4S service and the best-effort service classic service. The same goes for the respective AQM elements like for example queues or drop probabilities.

2.4.1 Basic Concept

The basic concept of L4S contains three elements [11]. These are separation, identification and scalable congestion control.

Separation means the separation of L4S traffic and classic traffic so that both of them do not affect each other. Mainly, the L4S traffic needs to be protected from the potentially high latency of the classic traffic. Also, the classic traffic needs to be protected from the less sensitive scalable congestion control of the L4S traffic. But even if the two classes are separated, they should still share the link capacity freely without having fixedly assigned shares.

Identification means the necessity to clearly identify the respective classes in order to correctly separate incoming traffic. A recommendation for this identifier is the ECT(1) code-point [25].

And scalable congestion control is necessary for the L4S traffic in order to avoid the problem of link utilization. Since the L4S traffic seeks low latency, only a small buffer can be used. And as mentioned in Section 2.2.3, when only a small buffer is available, scalable congestion control achieves higher utilization than classic congestion control. Also, scalable congestion control requires ECN which can prevent the large spikes in latency that occur when packets are lost. L4S also requires ECN since the ECT(1) code-point is recommended to be used as identifier. DCTCP is mentioned as a possible scalable congestion control.

2.4.2 DualQ Coupled AQM

DualQ Coupled AQM is a proposed implementation of L4S [26]. DualQ Coupled AQM uses the ECT(1) code-point to classify incoming packets. The traffic is separated into two queues, one for L4S and one for classic traffic. These queues run an AQM that is coupled through the drop probability. This coupling means that the drop probability for the classic Queue is equal to the scaled and squared drop probability of the L4S queue. Equation 2.10 shows this coupling.

\[
\text{drop probability}_C = \left( \frac{\text{drop probability}_L}{K} \right)^2
\]  

(2.10)

The squaring of the drop probability is necessary due to the different powers of the drop rate in the throughput equations of classic and scalable congestion control described in Section 2.2.3. With the scaling factor \( K \), this steady state distribution can be shifted. Two is recommended for \( K \) since it is a power of two and since it approximates throughput equivalence between
2.4 Low Latency Low Loss Scalable Throughput

L4S and classic flows. Using powers of two for $K$ has the advantage that divisions are cheaply implementable through bit shifts.

To achieve a low latency in the L4S queue, the L4S queue is to be scheduled with priority. This priority can be strict but does not have to be.

Two possible implementations of DualQ Coupled AQM are proposed. One uses PI2 as AQM and the other CRED.

Figure 2.8 shows the function principle of DualQ Coupled AQM.

DualQ Coupled AQM with PI2

This proposed implementation of DualQ Coupled AQM uses the AQM PI2 described in Section 2.3.4. It consists of an enqueuing function, a dequeuing function and a drop probability calculation function that periodically updates the drop probabilities.

The drop probabilities are calculated based on the current queuing delay. As long as the classic queue is not empty, the its queue delay is used for the calculation. If it is empty, the queue delay of the L4S queue is used. The new drop probabilities are calculated according to Equation 2.11.

\[ \text{classic\_drop\_probability} = \text{classic\_drop\_probability} + \alpha \times \text{update\_period} \times (\text{current\_queue\_delay} - \text{target\_queue\_delay}) \]
\[ + \beta \times \text{update\_period} \times (\text{current\_queue\_delay} - \text{old\_queue\_delay}) \]

\[ \text{l4s\_drop\_probability} = K \times \text{classic\_drop\_probability} \]

At enqueueing, the queues are checked whether there is space. If there is, the packet is classified using the ECT(1) code-point and enqueued in the respective queue.

For dequeueing, the queues are scheduled with a shifted FIFO scheduler with the shift in favour of the L4S queue. This means, the classic queue is only scheduled if the L4S queue is empty or if the oldest packet in the classic queue is more than a certain time older than the oldest in the L4S queue.

If an L4S packet is dequeued it is decided whether it will be marked. The packet is marked randomly with the L4S drop probability. If the packets has been in the queue longer than a time threshold and if the queue is currently longer than a length threshold, the packet is marked deterministically.

If a classic packet is dequeued it is randomly dropped or marked with the square of the classic drop probability.

This algorithm has an overload protection that would switch to dropping if it detected an unresponsive ECN flow. We will not explain this part since we did not examine the security properties of these algorithms in this work. The overload protection can be seen in the pseudo code in [26].

The current queue delay can either be measured by using timestamps or estimated using the current queue length and a measurement of the dequeuing rate. The proposed pseudo code uses timestamps.
These are recommended parameter values for the DualQ Coupled AQM with PI2 from [26].

- **Target delay**: 15 milliseconds
- **Update period**: 16 milliseconds
- **FIFO time shift**: $2 \times$ target delay (30 milliseconds)
- **Time threshold**: 1 millisecond
- **Length threshold**: 2 packets (or $2 \times$ MTU)
- **Maximum classic drop probability**: 0.25
- **Maximum L4S drop probability**: $\min(K \times \sqrt{\text{classic\_drop\_prob\_max}}, 1)$
- **Alpha**: 10
- **Beta**: 100
- **K**: 2

The complete, proposed pseudo code of DualQ Coupled AQM with PI2 can be seen in [26].

### DualQ Coupled AQM with CRED

This proposed implementation of DualQ Coupled AQM uses the AQM CRED described in Section 2.3.2. It only has an enqueuing and a dequeuing function.

The enqueuing here works like the enqueuing in DualQ Coupled AQM with PI2. The queues are checked whether they have free space and then the packet is sorted into the respective queue depending on the ECT(1) code-point.

For dequeuing, strict priority scheduling in favour of the L4S queue is used. If an L4S packet is dequeued, it is randomly marked with the L4S drop probability. And if the L4S queue exceeds a certain length threshold, the packet is marked deterministically.

If a classic packet is dequeued, it is randomly dropped or marked with the classic drop probability. If a packet from the classic queue is dropped, another one is dequeued until the classic queue is empty or one packet was not dropped.

The L4S drop probability and the classic drop probability are calculated according to Equation 2.13. For the calculation of the L4S drop probability, the current queue delay of the classic queue is used. For the calculation of the classic drop probability, an exponentially weighted moving average (EWMA) of the classic queue delay is used. This calculation can be seen in Equation 2.12.

The current queue delay can either be measured by using timestamps or estimated using the current queue length and a measurement of the dequeuing rate. The proposition does not specify which.

\[
Q_C = 2^{-\text{smoothing\_factor}} \times \text{classic\_queue\_delay} + (1 - 2^{-\text{smoothing\_factor}}) \times Q_C
\]  

(2.12)

\[
\text{classic\_drop\_probability} = \frac{Q_C}{2^{\text{classic\_scaling\_factor}}}
\]  

(2.13)

\[
\text{l4s\_drop\_probability} = \frac{\text{classic\_queue\_delay}}{2^{\text{l4s\_scaling\_factor}}}
\]

We assume there is a typo or a sign error in the draft. The formula for the L4S drop probability in Equation 2.13 seems incorrect since it does not conform to the throughput equivalence equation in Section 2.4.2. This is discussed further in Section 3.1.3.

These are recommended parameter values for the DualQ Coupled AQM with CRED from [26].
The Network Simulator 3 (NS-3) is an open source software for network simulations [1]. We used this software to simulate a router handling congestions with different AQM schemes.

### 2.5.1 Direct Code Execution

Direct Code Execution (DCE) is a feature of NS-3 [2]. DCE allows a user to run real-world kernel code libraries on the simulated devices. Using original kernel code allows the use of the real world protocol implementations. This should give a more realistic behaviour of the different algorithms in the simulation.

For our simulations we used the linux kernel version 4.7.0 -rc5 [4].

### 2.5.2 Link Models

In NS-3, different link models can be used for simulations. With these models, different kinds of networks can be simulated. For our simulations we used the static link model and the LTE link model.

**Static Link Model**

The static link model simulates a point-to-point link that is statically defined by a bandwidth and a delay. [1]

**LTE Model**

The LTE model allows simulations of an LTE network [3]. For our simulations, the important aspects of the model are the bearers and the bearer scheduling.

In LTE, there are so-called bearers. Simplified, bearers are virtual channels. For every user device in an LTE network a default bearer is opened but additional dedicated bearers can be added. Dedicated bearers can be used to separate specific traffic like for example voice over IP.

The LTE model of NS-3 simulates this bearer system and allows a user to add his own bearers depending on what he wants to simulate. Each bearer maintains its own queue.

Bearers have to be scheduled for transmission by the LTE network. NS-3 has different schedulers implemented. For our simulations, we used the proportional fair scheduler. The proportional fair scheduler calculates a priority for each bearer according to Equation 2.14 where $R_i$ is the achievable throughput of bearer $i$ in the coming time slot and $T_i$ is the achieved throughput in the past of bearer $i$. $T_i$ is calculated using an exponentially weighted moving average.

The bearers are scheduled with these calculated priorities.
2.6 Jain’s Fairness Index

Jain’s fairness index is a measure for how fair bandwidth was distributed between multiple flows [18]. It is calculated with from the throughputs achieved by the individual flows. The calculation can be seen in Equation 2.15 where $n$ is the total number of flows and $t_i$ is the throughput of flow $i$.

$$jain\_fairness\_index = \frac{\left(\sum_{i=0}^{n} t_i\right)^2}{n \times (\sum_{i=0}^{n} t_i^2)}$$  \hspace{1cm} (2.15)

Jain’s fairness index is bounded between 0 and 1. The distribution was absolutely fair if the index equals one.
Chapter 3
Simulation

3.1 AQM Implementations

For our simulations, we implemented different algorithms and AQM schemes in order to evaluate their performance in different scenarios. We implemented three groups of algorithms. The simplest implementations use a single queue with an AQM. Their results should show how established algorithms perform in our simulations and serve as a reference for the other implementations. Some more complex algorithms use two queues where low-latency and classic traffic can be separated. Both queues run independent AQM schemes without coupling. The two queues are scheduled with a weighted round robin scheduler. These implementations are simplifications of the proposed DualQ Coupled AQM described in Section 2.4.2. For one, their results should show the change in performance between separated treatment and Single Queue AQM implementations. And furthermore, these results should show how the additional step of coupling and priority scheduling does further change the performance. The most complex implementations are the DualQ Coupled AQM implementations described in Section 2.4.2. Based on these results we want to evaluate the performance of this proposed algorithm.

3.1.1 Single Queue AQM

The Single Queue AQM implementations work like a classic buffer. Incoming packets are stored in a FIFO queue until they can be passed on. Packets are treated equally independent of their traffic class. We implemented the Single Queue AQM with RED and PIE described in Section 2.3.1 and 2.3.3.

In early simulations we saw that DualQ Coupled AQM could keep queuing delays in the order of a few milliseconds. Since the Single Queue AQM implementations would have to compete with that, we decided to use a target delay of five milliseconds.

Since these implementations do not separate low-latency and classic flows, the Single Queue AQM implementations can only work fairly if all flows use either classic or scalable congestion controls (See Section 2.2.3). And since current traffic in the internet uses classic congestion controls, the low-latency flows must do that as well. Therefore, classic congestion controls were used on all flows in the simulations with Single Queue AQM.

Single Queue AQM with RED

NS-3 already contains an implementation of RED but we implemented our own. The reasons were that the out-of-the-box implementation did not support ECN and that we wanted to have an exact implementation of RED as it is in [12]. We implemented RED according to that description.
The parameters for RED were set according to the recommendations in [12] and our own experimental simulations. The maximum drop probability was set to 0.02 or 2% as recommended. Since RED does not take a target delay as parameter, we experimentally determined the thresholds that would correspond to the target of five milliseconds. To reduce the number of parameters, we decided to use a minimum threshold of one third the maximum threshold. The experimental simulations were made for a single Reno-controlled flow and the link speed used in the simulations. This resulted in a maximum threshold of 34 packets. Dividing by three resulted in a minimum threshold of 11 packets.

**Single Queue AQM with PIE**

NS-3 also already contains an implementation of PIE but again we implemented our own. Again, the reasons were that the included NS-3 implementation did not support ECN and that we wanted an exact implementation according to [22]. Our implementation of PIE followed that description. For our PIE implementation, we used timestamps to determine the queue delay and not an estimation through dequencing rate and queue length.

The parameters for PIE were set according to the recommendations in [22] except for the target queue delay and the update period of the drop probability. Like for Single Queue RED, the target queue delay was set to five milliseconds and the update period as well. The update period was reduced since the recommended one would have been higher than the target delay. Early simulations showed that PIE could keep the target better with a reduced update period. The update period was set equal to the target delay since it is equal to the target delay in [22] as well.

**3.1.2 Dual Queue Uncoupled AQM**

The Dual Queue Uncoupled AQM implementations work with two FIFO queues with one for the classic traffic and one for the low-latency traffic. Like in [26], we use the ECT(1) code-point as a class identifier. Incoming packets were separated into the two queues based on the ECT(1) code-point. The two queues were scheduled with a weighted round robin scheduler. The weight was assigned according to the number of flows in each class. The flow numbers are not determined by the queues themselves but given to them as an argument. An actual implementation would have to determine the numbers by itself. For the Dual Queue Uncoupled AQM implementations we also used the AQM schemes RED and PIE described in Sections 2.3.1 and 2.3.3.

Both queues ran independent instances of the same AQM. For the low-latency queue, a low target queue delay was set in order to serve the low-latency requirement. Again, since the AQM would have to compete with DualQ Coupled AQM, this delay was set to five milliseconds. The classic queue AQM was set up with a high target queue delay in order to achieve high throughput. Also, we saw in early simulations that a high classic target delay improved the performance of the low-latency AQM. When the target delay of the classic queue AQM was set too low, the classic queue often got empty in which case the low-latency queue received more throughput. Once the low-latency queue received more throughput, the low-latency senders increased their sending rate. When the classic queue then filled again and reclaimed the throughput, the increased sending rate leading to longer queue and increased delay. Therefore, the classic target delay was set so high that the classic queue delay should never run empty.

Since these implementations did separate low-latency and classic traffic, scalable congestion control could be used on the low-latency flows. Since the throughput was distributed by the scheduler, the scalable congestion control flows in the low-latency queue could not starve the classic congestion control flows in the classic queue. So classic and scalable congestion
control could coexist.

**Dual Queue Uncoupled AQM with RED**

For Dual Queue Uncoupled AQM with RED, we again used the basic RED implementation from [12]. The difference was that here we ran two instances of the algorithm in parallel. Since RED does not hold a state, the only difference between the instances were the parameters.

Like in the Single Queue AQM implementations, the parameters were set according to the recommendations in [19] and our own experimental simulations. For both instances, the maximum drop probability was set to 0.02 or 2% as recommended. The minimum and the maximum threshold for the low-latency instance were again determined experimentally. And like in Single Queue RED, we used a ratio of 1:3 between minimum and maximum threshold to simplify the determination. Since the average queue length in RED is different for scalable and classic congestion control, we ran additional simulations for a single DCTCP controlled flow. Based on these simulations we set the thresholds corresponding to a target delay of five milliseconds.

The base minimum and maximum thresholds for classic congestion controls were like in the Single Queue AQM implementation 11 and 34 packets. The simulations showed that the base maximum threshold for scalable congestion control should be five packets and the minimum threshold therefore two packets.

The weighted round robin scheduler assigned throughput according to the flow counts and the delays do also depend on the throughput beside the queue length. Therefore, the resulting values were scaled in each simulation for the expected throughput of the low-latency traffic. Equation 3.1 shows the scaling.

\[
\text{max\_thresh\_low\_latency} = \max\left(\frac{\text{low\_latency\_flows}}{\text{low\_latency\_flows + classic\_flows}} \times \text{base\_max\_thresh}, 1\right)
\]

\[
\text{min\_thresh\_low\_latency} = \min\left(\frac{\text{max\_thresh\_low\_latency}}{3}, \text{max\_thresh\_low\_latency} - 1\right)
\]

The minimum and the maximum threshold for the classic AQM instance needed to be set to high values in order to keep the throughput high and the classic queue full. Therefore, the minimum and maximum threshold were set to 100 and 300 packets.

**Dual Queue Uncoupled AQM with PIE**

For Dual Queue Uncoupled AQM with PIE, we again used the same basic PIE implementation described in Section 2.3.3. Again, two instances of the algorithm were run in parallel. Here, the difference included state variables aside from the parameters.

Like in the Single Queue AQM implementation, the parameters were set according to the recommendations from [22] with the exception of the target queue delay and the update period. For the low-latency instance, the target delay and the update period was again set to five milliseconds.

The target delay of the classic AQM instance was set to 500 milliseconds to achieve high throughput and keep the classic queue full. For the classic instance, the update period was taken from the recommendations in [22] since it was still lower than the target delay.

### 3.1.3 DualQ Coupled AQM

DualQ Coupled AQM was implemented as described in Section 2.4.2. We implemented three versions of this algorithm.
Proposed Implementations

[26] contains two proposed implementations for DualQ Coupled AQM. One is using PI2 and the other is using CRED.

The proposed implementation using PI2 is a fully elaborated algorithm that we implemented according to the pseudo code from [26].

The proposition using CRED is not as elaborate. Our implementation follows the pseudo code from [26] aside from three points.
First, the proposed pseudo code does not allow to use ECN marking in the classic queue. For our implementation we decided to add this option.
Second, the proposition mentions two mechanism to determine the queue delays. One is the use of timestamps, the other is a delay estimation based on the queue length and the dequeuing rate. But it is not specified, which one should be used. We decided to use timestamps.
And third, we think that there is an error in the pseudo code concerning the drop probability calculation. The drop probability calculation from [26] does not satisfy the throughput equivalence equation described Section 2.4.2. This drop probability calculation is shown in Equation 3.2.

\[
\text{classic\_drop\_probability} = \left( \frac{Q_C}{2^{\text{classic\_scaling\_factor}}} \right)^2 
\]

\[
l4s\_drop\_probability = \frac{\text{classic\_queue\_delay}}{2^{l4s\_scaling\_factor}} 
\] (3.2)

In steady state, the exponentially weighted moving average \( Q_C \) should approximate the current queuing delay. And according to [26], \( l4s\_scaling\_factor = \text{classic\_scaling\_factor} + \log_2 K \).

This results in the drop probabilities and coupling shown in Equation 3.3.

\[
\text{classic\_drop\_probability} = \left( \frac{\text{classic\_queue\_delay}}{2^{\text{classic\_scaling\_factor}}} \right)^2 
\]

\[
l4s\_drop\_probability = \frac{\text{classic\_queue\_delay}}{2^{\text{classic\_scaling\_factor} + \log_2 K}} 
\] (3.3)

\[
\Rightarrow \text{classic\_drop\_probability} = (l4s\_drop\_probability \times K)^2 
\]

In order for the drop probabilities to satisfy the equivalence equation, the L4S scaling factor needs to be \( \text{classic\_scaling\_factor} - \log_2 K \) instead of \( \text{classic\_scaling\_factor} + \log_2 K \). We assume this is a sign error in [26].

For our implementation, we changed the calculation to satisfy the equivalence equation.

DualQ Coupled AQM with PIE

Additionally to the two proposed implementations, we implemented a third version based on the PIE algorithm described in 2.3.3. The basic design was still the one of DualQ Coupled AQM from with two queues, priority scheduling, identifier and coupling.

For this implementations we used the unchanged drop probability calculation function of PIE described in Section 2.3.3.
The enqueuing function was changed to be able to serve two queues. After checking whether the queues still have free space, the incoming packets were classified based on the ECT(1) code-point. After the classification, the drop decision was made. Classic packets were dropped or marked according to the classic PIE scheme. L4S packets were marked randomly with the L4S drop probability which was equal to the square root of the classic drop probability times the scaling factor. Equation 3.4 shows the calculation of the L4S drop probability. Also, L4S packets were marked deterministically if the queue delay and the queue length exceeded certain thresholds. This deterministic marking and its thresholds were borrowed from the proposed implementation with PI2 in [26].

\[ l4s_{\text{drop probability}} = \sqrt{\text{classic\_drop\_probability}} \times K \]  

(3.4)

If the incoming packet was not dropped, it was enqueued in the respective queue.

The dequeuing function implemented a strict priority scheduling in favour of the L4S queue.

The queue delays were determined using timestamps.

### 3.2 Kernel Modifications

For our simulations we used the Linux kernel version 4.7.0-rc5 [4]. For the purpose of this simulation, we made some modifications to the kernel code.

The first modification concerns the ECN behaviour. We changed the kernel code to use ECT(1) instead of ECT(0) in order to mark traffic as low-latency traffic. This modification was used for the low-latency and L4S senders.

The second modification concerns the congestion control. DCTCP is already implemented in [4]. But since we wanted to compare the AQM schemes with another scalable congestion control, we additionally implemented Relentless [21]. Our implementation was based on the existing DCTCP implementation [4]. Most of the DCTCP code was left as it is. The reduction function was changed to reduce the congestion window by half a packet for every received ECN feedback. If an ECN feedback acknowledged multiple packets, the reduction was increased accordingly.

[21] recommends a reduction by one packet for every received ECN feedback but we wanted to tune Relentless to approximately the same sensitivity as DCTCP. DCTCP halves its congestion window when all packets in a round-trip-time are marked [7]. With a reduction of half a packet per ECN feedback, Relentless is similarly sensitive.

### 3.3 Simulation Network

#### 3.3.1 Static Link Model

The first simulations used the static link model described in Section 2.5.2 for the simulation network. These simulations should provide insight into of the performance that these AQM schemes would have in a wired network. We set up our simulation network with bandwidths of 1 gigabit per second on the access links and 10 megabits per second on the congested link. The base round-trip-time, was set to 100 milliseconds.

For the simulations with the static link model, we implemented all the AQM schemes mentioned in Section 3.1.
Figure 3.1: Static link simulation network

Figure 3.1 shows the static link network used for the simulations.

3.3.2 LTE Network

The second simulations used the LTE link model described in Section 2.5.2 for the congested link. These simulations should demonstrate the performance of the AQM schemes in a mobile network.

For these simulations, the parameters of the congested link were defined by the LTE model of NS-3. We saw that the maximum bandwidth was around 17.5 megabits per second. We also saw that the base round-trip-time of the LTE link was in the order of a few milliseconds.

The access links to the LTE base station were set up with a bandwidth of 1 gigabit per second and a base round-trip-time of 100 milliseconds.

For these simulations, we implemented the Single Queue and Dual Queue Uncoupled AQM schemes described in Sections 3.1.1 and 3.1.2. From the DualQ Coupled AQM schemes described in Section 3.1.3, we could only implement the version with PIE. These implementations were more complicated since they had to be written inside the code of the NS-3 LTE module [3] itself.

This imposed some restrictions. For example, we could only implement AQM schemes that work at enqueuing. The dequeuing functions were more complex since they did not simply dequeue packet by packet but also grouped them into transmission units. If we wanted to include AQM schemes in these functions, we would also have had to make sure that these AQM schemes do not interfere with this grouping. This would have significantly added complexity. Ultimately, we did not have enough time to do this.

We also used the bearer scheduling mechanisms that were already implemented in the LTE module [3]. Due to time limits, we did not look closely into these schedulers. They were quite complex since they had to cooperate with other elements of the LTE model. This made it more complicated to implement our own schedulers and we did not have enough time to do this. Therefore, we used the included proportional fair scheduler described in Section 2.5.2 instead of the weighted round robin or priority schedulers.

Figure 3.2 shows the LTE network used for the simulations.

3.4 Simulation Scenarios

For our simulation scenarios we used several parameters. The most important ones were the flow counts in each class. Additionally we used different congestion controls and turned ECN on and off for the classic flows.
The flow configurations we used were all combination of one, five and ten flows in both traffic classes. These simulations should show how the AQM schemes perform with different flow counts per class and different ratios of flow counts between the classes.

We also ran the simulations with different combinations of congestion controls in both the low-latency senders and the classic senders. For the low-latency senders Reno, Cubic, DCTCP and Relentless were used depending on what was sensible to be used for the current AQM. For the classic senders, Reno and Cubic were used. These different simulations should show whether the choice of congestion control has a significant impact on the performance of the AQM schemes.

Also, we ran the simulations with ECN activated and deactivated for the classic flows. ECN was always activated for the low-latency flows because the ECT(1) code-point was necessary for the traffic class separation and because ECN was necessary for the scalable congestion controls. These simulations should show, whether the using ECN on the classic flows has an impact on the performance of the AQM schemes.

In all our simulations we used a segment size of 1502 bytes.
Chapter 4

Results

4.1 Metrics

For evaluating the results of our simulations we used metrics concerning the delays and the throughput.

We evaluated the delays by looking at the maximum and the average queue delay. For the implementations with two queues, this was extended to maximum and average queue delays of the classic and the low-latency or L4S queue. Since only the low-latency or L4S traffic seek low latency, the main metric in the implementations with two queues were the maximum and average low-latency or L4S queue delays.

The throughput was evaluated by looking at the link utilization and the fairness. The utilization should show how much resources were wasted by a certain algorithm. The fairness should show how fairly the available bandwidth was distributed between the two traffic classes. In order to measure the fairness, we used Jain’s fairness index described in Section 2.6. For the calculation of the index we assumed that within the traffic classes, the throughput was assigned fairly.

For our purposes we modified the fairness index. In Jain’s classic fairness index, it is not shown, which class got more throughput in the case of unfairness. Therefore, we changed the index depending on which class received more throughput.

When the classic traffic class received more throughput, the fairness index was left as it is. When the low-latency or L4S traffic class received more throughput, the sign of the index was flipped and two was added to it. This modified index is still the fairest when it is close to one. But it shows now which class received more throughput depending if it is below or above one. Equation 4.1 shows the calculation performed when the low-latency or L4S class received more throughput.

\[
\text{modified\_jain\_index} = (2 - \text{jain\_index})
\]  

(4.1)

4.2 Static Link Simulations

This section discusses the results from our simulations in the static link network.

In the following passages, we only use examples to visualize our findings. The full data and all plots can be found in Appendices A.1 and B.1.
4.2.1 Single Queue AQM

In our simulations, we saw that Single Queue PIE achieved the target delay of 5 milliseconds only for two total flows. For more flows, the average queue delay increased, up to a maximum of 8.8 milliseconds. Single Queue RED missed the target delay for all flow combinations. The lowest average queue delays were achieved with only two flows, having been between 7.8 and 11 milliseconds. For more flows, the average queue delays went up to between 18 and 28 milliseconds.

Like the average queue delays, the maximum queue delays in both single queue AQM were the lowest for two flows and went up for more flows. And like for the average queue delay, PIE achieved lower maximum queue delays than RED.

We think the reason for this is that the target delay of five milliseconds translates to a too small queue size. With a segment size of 1502 bytes and a bandwidth of 10 megabits per second, one packet takes about 1.2 milliseconds to transmit. This means that a queue of more than four packets already exceeds the target delay. This gives the AQM too little space to work, especially for higher flow counts.

By trying to reach such a low delay, the single queue AQM often underutilized the link. RED was loosing up to 14.5% utilization and PIE up to 17.5%.

In most cases, the available bandwidth was shared fairly between the flows with the exception of ECN unfairness (See Section 2.1) that occurred when using RED.

Influence of Flow Counts

In general, the queue delays in Single Queue AQM schemes increased with the number of flows. The main increase in delay was from two to six flows. Above six flows, the increase slowed down.

Also, ECN unfairness in RED became more severe with increasing classic flow counts. We think, this comes from the increasing size and frequency of spikes due to more flows. These spikes would cause the queue to exceed the maximum threshold (See Section 2.3.1). In that case, the drop probability would be 100% leading to more severe ECN unfairness. Activating ECN also lead to an increase in maximum queue delay. We think this is because unlike dropped packets, marked packets were enqueued anyway, adding to the queue length and delay.

The Figures 4.1 and 4.2 show the resulting metrics from simulations with Single Queue AQM across all flow count combinations. These plots should visualize the influence of the flow counts. We chose the results from the simulations with Cubic as examples. The results from the simulations with Reno showed similar effects.

Influence of TCP Settings

When ECN was deactivated on the classic links, single queue RED encountered ECN unfairness. Activating ECN also generally lead to an increase in maximum delay since the packets that would be dropped without ECN are marked and enqueued anyway, increasing the delay.

Using Cubic instead of Reno as congestion control decreased the maximum delays and increased the utilization.

We think the reason for this was the different growth function of cubic. Since the bandwidth was quite stable, the Cubic algorithm (See Section 2.2.1) only rarely reached the phase of fast, exponential growth. Therefore, the congestion window of Cubic was growing slower around the saturation point than that of Reno. This lead to lower peaks in queue length and queue delay. Also, Cubics initial fast growth in the concave phase brought up the congestion window faster than the linear growth of Reno leading to higher utilization.

The Figures 4.3a and 4.3b show examples of congestion window traces from our single queue AQM.
4.2 Static Link Simulations

Figure 4.1: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the Single Queue AQM simulations using Cubic on all flows and activating ECN on the classic flows.

Figure 4.2: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the Single Queue AQM simulations using Cubic on all flows and deactivating ECN on the classic flows.
It shows the aforementioned differences between Cubic and Reno.

Figure 4.4 shows the resulting metrics from simulations with Single Queue AQM across all available TCP settings. These plots should visualize the influence of the TCP settings. We chose the results from the simulations with 5 low-latency and 5 classic flows as an example. The results from the simulations with other flow configurations showed similar effects.

### 4.2.2 Dual Queue Uncoupled AQM

Unlike in Single Queue AQM, scalable congestion control could be used for the low-latency flows in Dual Queue Uncoupled AQM. In our simulations with Dual Queue Uncoupled AQM we also looked at the difference that scalable congestion control can make.

With classic congestion control, the PIE implementations also only achieved the low-latency target delay of 5 milliseconds when there were only 2 flows. For more flows, the average low-latency queue delay increased, up to a maximum of 23 milliseconds. The RED implementation could never keep the low-latency target delay. With only two flows, the average low-latency queue delay was the lowest between 8.3 and 12 milliseconds. For more flows, the average low-latency queue delay increased to between 18 and 25 milliseconds. For PIE and RED, the maximum low-latency queue delays were also the lowest for only two flows and increased for more.

With scalable congestion control, the average low-latency queue delays of PIE were very similar to the ones with classic congestion control. But the maximum low-latency queue delay tended to be smaller with scalable than with classic congestion control. The low-latency queue delays in the RED implementation were significantly smaller when using scalable congestion control. Now, RED was sometimes able to keep the low-latency target delay, especially for low flow counts. But the RED implementation still had cases where the average low-latency queue delay went up to 17 milliseconds. And like in the PIE implementations, scalable congestion control resulted in lower maximum low-latency queue delays than classic congestion control in Single Queue RED.

The average and maximum queue delays of the classic queue were in the order of hundreds or sometimes thousands of milliseconds for both PIE and RED as well as classic and scalable congestion control. These high delays came by design. The AQM of the classic queue has a very high target delay in order to keep it at a high filling level. This high filling level should keep the classic queue from running empty, allowing it to use all available bandwidth, serving its goal...
4.2 Static Link Simulations

Figure 4.4: Average and maximum queue delays, link utilization and modified Jain index over all available TCP settings for the Single Queue AQM simulations with 5 low-latency and 5 classic flows of high throughput.

By design, both implementations achieved almost full link utilization for classic and scalable congestion control. The lowest link utilization was 99.88%. Dual Queue Uncoupled AQM was designed to achieve full link utilization by keeping the classic queue from running empty. So when the low-latency queue had to give up bandwidth in order to achieve its target delay, the classic queue could absorb it. Because of that, the important metric concerning throughput was the allocation. With both scalable and classic congestion control, this allocation was fair in most cases.

Influence of Flow Counts
The low-latency queue delays generally increased with the number of flows. Additionally, the low-latency delays increased when there were more classic than low-latency flows. We think, this is an effect of the weighted round robin scheduler. When the low-latency class had less flows than the classic class, then it also got scheduled less often. This lead to longer intervals between dequeues and therefore higher delays.

In cases where there were more low-latency than classic flows, the low-latency queue delay was generally the lowest. But in these cases, the low-latency queue was often also not able to fully use the assigned throughput. Since this bandwidth was not lost but absorbed by the classic queue, this did not lead to a lower utilization but to unfairness. We think the reason for this is that with higher numbers of flows, the low-latency class received more throughput which made it easier to achieve its target delay. But often, when the target delay went down the fairness did as well. Also, the choice of congestion control had a significant influence on this.

Figure 4.5 shows the resulting metrics from simulations with Dual Queue Uncoupled AQM across all flow count combinations when using classic congestion controls on the low-latency
36 CHAPTER 4. RESULTS

Figure 4.5: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the Dual Queue Uncoupled AQM simulations using Cubic on all flows and activating ECN on the classic flows.

Influence of TCP Settings

When using classic congestion control, both RED and PIE showed the aforementioned problem of unfairness. With Reno, this unfairness was in most cases more severe than with Cubic. And Cubic also tended to achieve lower maximum low-latency queue delays.

We think, the reason for this is the same as in the Single Queue AQM.

Figures 4.7a and 4.7b show examples of low-latency sender congestion window traces from our Dual Queue Uncoupled RED simulations using Reno and Cubic. Like in the Single Queue AQM case, the differences leading to lower maximum delay and higher utilization can be seen in these plots.

When using scalable congestion control, the aforementioned problem with unfairness persists for RED when Relentless is used. For PIE and for RED with DCTCP, the bandwidth distribution is always very fair.

The unfairness in RED with Relentless was in some cases even more severe than with classic congestion controls. Apart from the difference in fairness, Relentless also reached lower average low-latency queue delays.

We think the reason for this is that neither RED nor Relentless contain any smoothing function. This lead to more fluctuations in the congestion window. With the limited buffer size due to delay constraints this caused the low-latency queue to give up more throughput. This also reduced the average queue length and therefore the average low-latency queue delay.

Figures 4.8a and 4.8b show examples of low-latency sender congestion window traces from our Dual Queue Uncoupled RED simulations using DCTCP and Relentless. It can be seen that
4.2 Static Link Simulations

Figure 4.6: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the Dual Queue Uncoupled AQM simulations using DCTCP on low-latency flows, Cubic on classic flows and activating ECN on the classic flows.

(a) Low-latency sender congestion window trace of a simulation with Reno

(b) Low-latency sender congestion window trace of a simulation with Cubic

Figure 4.7: Comparison of the congestion windows of Reno and Cubic from our Dual Queue Uncoupled RED simulations.
CHAPTER 4. RESULTS

(a) Low-latency sender congestion window trace of a simulation with DCTCP
(b) Low-latency sender congestion window trace of a simulation with Relentless

Figure 4.8: Comparison of the congestion windows of DCTCP and Relentless from our Dual Queue Uncoupled RED simulations

the congestion window trace of Relentless is much less smooth than the one of DCTCP.

When scalable congestion control was used on the low-latency flows, the choice of classic congestion control on the classic flows did not have a significant effect on the low-latency delays or the fairness. This makes sense since these congestion controls were only used to control the classic queue which is kept at a high filling level anyway.

Figure 4.9 shows the resulting metrics for simulations with Dual Queue Uncoupled AQM across all available TCP settings when using classic congestion controls on the low-latency flows. Figure 4.10 shows the same for when using scalable congestion controls on the low-latency flows. These plots should visualize influence of the TCP settings. We chose the results from the simulations with 5 low-latency and 5 classic flows as examples. The results from the simulations with other flow configurations showed similar effects.

4.2.3 DualQ Coupled AQM

In our simulations, DualQ Coupled AQM achieved very low delays in the L4S queue. The average L4S queue delays were between 1.0 and 3.1 milliseconds for CRED, between 1.7 and 3.3 milliseconds for PI2 and between 1.1 and 2.1 milliseconds for PIE. The maximum L4S queue delays were between 4.5 and 19 milliseconds for CRED, between 10 and 35 milliseconds for PI2 and between 4.5 and 14 milliseconds for PIE.

While all implementations achieved similarly low average L4S queue delays. The PI2 implementation tended to reach higher maximum L4S queue delays. We think this is due to the scheduler. While the scheduler for CRED and PIE were strict priority schedulers, PI2 used a shifted FIFO scheduler (See Sections 2.4.2 and 3.1.3). With this non-strict priority scheduler, the L4S queue could sometimes lose its priority leading to higher maximum delays.

The classic queue delays in the DualQ Coupled AQM simulations were lower than the ones in the Dual Queue Uncoupled AQM simulations. The average classic queue delays were between 7.6 and 86 milliseconds for CRED, between 15 and 19 milliseconds for PIE and between 16 and 36 milliseconds for PIE. The maximum classic queue delays were between 37 and 350 milliseconds for CRED, between 39 and 62 milliseconds for PIE and between 50 and 325 milliseconds for PIE.

Note again that, the CRED and PIE implementations reached similar delays in the classic queue but the PI2 implementation did not. Like for the L4S queue delays, we think the reason for this was the non-strict priority scheduling of PI2 that sometimes suspended the priority of the L4S queue.
4.2 Static Link Simulations

Figure 4.9: Average and maximum queue delays, link utilization and modified Jain index over all available TCP settings for the Dual Queue Uncoupled AQM simulations with 5 low-latency and 5 classic flows when using classic congestion control on all flows.

Figure 4.10: Average and maximum queue delays, link utilization and modified Jain index over all available TCP settings for the Dual Queue Uncoupled AQM simulations with 5 low-latency and 5 classic flows when using scalable congestion control on low-latency flows on low-latency flows.
In the simulations, DualQ Coupled AQM experienced losses of link utilization up to 12%. The highest link utilization was generally achieved by CRED, followed by PI2 and PIE. The highest losses were seen when there was only one classic flow. This makes sense since with only one flow, the AQM is most likely to cause the classic queue to run empty.

All implementations experienced some problems in terms of fairness. In a lot of cases, the L4S queue achieved less throughput than the classic queue. We think, one general reason for this is the choice of the factor $K$ that scales the drop probabilities in the coupling. [26] recommends 2 since multiplication and division by powers of 2 is cheaply implementable using bit shifts. But according to the throughput equations described in Section 2.2.3, the scaling factor with DCTCP should be 1.22 for Reno and 1.68 for Cubic. This puts the L4S queue at a disadvantage.

And like with Single Queue RED, ECN unfairness occurred with the CRED implementation.

### Influence of Flow Counts

With increasing L4S flow counts, all implementations encountered increasing L4S queue delays.

In the PIE and CRED implementations, the L4S queue delays tended to decrease with increasing classic flow counts. We think, the reason for this is that the AQM which depended on the classic queue had to work more aggressively for more flows leading to tighter control on the L4S queue.

In the PI2 implementation, the L4S queue delays tended to increase with the classic flow count. We also saw that more classic flows also increased the maximum classic queue delay. We think, this caused the L4S queue to lose the priority more often since DualQ Coupled PI2 used a shifted FIFO scheduler.

Table 4.1 shows the maximum classic queue delays from the simulations with DualQ Coupled PI2 when DCTCP was used on the L4S flows and ECN was activated on the classic flows. DCTCP and activated ECN was chosen as an example. The maximum classic queue delays were similar when using Relentless or when deactivating ECN on the classic flows. In can be seen that the maximum classic queue delay increases with the classic flow count.

For unbalanced flow counts, the fairness tended to decrease at the expense of the class with more flows. For CRED, this effect was seen the least. It was seen a bit stronger for PI2 and strongest for PIE.

Also, ECN unfairness became more severe with increasing classic flow counts. Like in Single Queue RED, we think this comes from increasing size and frequency of spikes in the classic queue length. These spikes would cause the queue delay to exceed the slope factor (See Section 2.3.2). In that case, the drop probability would be 100% leading to more severe ECN unfairness.

Figures 4.11 and 4.12 show the resulting metrics from simulations with DualQ Coupled across all flow count combinations. These plots should visualize the influence of the flow counts. We chose the results from the simulations with DCTCP and Cubic as examples. The results from

<table>
<thead>
<tr>
<th>Classic Congestion Control</th>
<th>Flow Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L4S Flows : Classic Flows</td>
</tr>
<tr>
<td></td>
<td>1:1</td>
</tr>
<tr>
<td>Cubic</td>
<td>39</td>
</tr>
<tr>
<td>Reno</td>
<td>41</td>
</tr>
</tbody>
</table>

Table 4.1: Maximum classic queue delay in milliseconds from the DualQ Coupled PI2 simulations using DCTCP and activating ECN on the classic flows.
4.2 Static Link Simulations

Figure 4.11: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the DualQ Coupled AQM simulations using DCTCP on L4S flows, Cubic on classic flows and activating ECN on the classic flows.

the simulations with other congestion controls showed similar effects.

**Influence of TCP Settings**

When ECN was deactivated on the classic links, the implementations with CRED encountered ECN unfairness (See Section 2.1).

In PI2 and for larger classic flow counts, using Cubic instead of Reno on the classic flows tended to result in lower maximum L4S queue delays. We saw that the in these cases, the maximum classic queue delay was also lower. And since the PI2 implementations uses a shifted FIFO scheduler, we assume that the L4S queue less often lost the priority when classic queue delays were lower.

We think the reason for the lower classic queue delays was the same as in Single Queue and Dual Queue Uncoupled AQM cases.

Table 4.1 shows the maximum classic queue delays from the the simulations with DualQ Coupled PI2 when DCTCP was used on the L4S flows and ECN was activated on the classic flows. DCTCP and activated ECN was chosen as an example. The maximum classic queue delays were similar when using Relentless or when deactivating ECN on the classic flows.

Just as for Dual Queue Uncoupled AQM, Relentless also achieved lower delays than DCTCP. But Relentless also gave up more throughput. Like in the Dual Queue Uncoupled AQM cases, we think that the reason is the lack of a smoothing function leading to large fluctuations in the congestion window.

Figures 4.13a and 4.13b show examples of low-latency sender congestion window traces from our DualQ Coupled PI2 simulations using DCTCP and Relentless. It can be seen that the congestion window trace of DCTCP is smoother than the one of Relentless.

Figure 4.14 shows the resulting metrics from simulations with DualQ Coupled AQM across all available TCP settings. These plots should visualize the influence of the TCP settings. We
Figure 4.12: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the DualQ Coupled AQM simulations using DCTCP on L4S flows, Cubic on classic flows and deactivating ECN on the classic flows.

Figure 4.13: Comparison of the congestion windows of DCTCP and Relentless from our DualQ Coupled PI2 simulations.
chose the results from the simulations with 5 low-latency and 5 classic flows as an example. The results from the simulations with other flow configurations showed similar effects.

### 4.2.4 Comparison

In our simulations we saw that Single Queue AQM could reach a low target delay only for low flow counts. And by trying to achieve a low delay, they sacrificed link utilization. Also, the ECN unfairness that was discovered, poses a problem for the use of ECN in Single Queue RED.

Dual Queue Uncoupled AQM also often exceeded the target delay when using classic congestion control on the low-latency flows. Compared to the Single Queue AQM, Dual Queue Uncoupled AQM in some cases even had higher delays, especially in cases with more classic than low-latency flows. As stated above, we think this is an inherent problem of the scheduler. Switching to scalable congestion control reduced the delays in Dual Queue Uncoupled AQM, especially for the RED implementation. But in cases with more classic than low-latency flows, the problem with the scheduler persisted.

The delays of the classic queue were quite high, in the order of hundreds and sometimes thousands of milliseconds. This was to be expected due to the design of Dual Queue Uncoupled AQM.

But, unlike the Single Queue AQM, Dual Queue Uncoupled AQM achieved almost full link utilization. Here, the problem concerning throughput was the fairness. Like in the Single Queue AQM, once the low-latency queue achieved a lower delay, it had to give up throughput. This throughput was absorbed by the classic queue and lead to unfairness.

Some scalable congestions control partially solved this problem as they should (See Section 2.2.3). With DCTCP, the bandwidth allocation was quite fair in all cases. Relentless also achieved a quite fair allocation with PIE. But with RED, Relentless in some cases even deteriorated the fairness.
The DualQ Coupled implementations were able to achieve very low L4S queue delays. Its L4S queue delays were lower than the delays in Single Queue AQM or the low-latency queue delays in Dual Queue Uncoupled AQM. Also, the delays of the classic queue were lower in DualQ Coupled AQM than the ones in Dual Queue Uncoupled AQM. But DualQ Coupled AQM had to sacrifice link utilization, unlike Dual Queue Uncoupled AQM. And like in the Single Queue RED, the ECN unfairness spotted in DualQ Coupled CRED poses a problem for the use of ECN. But unlike in Single Queue AQM, the use of ECN is necessary for DualQ Coupled AQM.

4.3 LTE Link Simulations

In this section we discuss the results from the simulations in the LTE network. Due to implementation restrictions, a proportional fair scheduler was used on all implementations with two queues. Since DualQ Coupled AQM requires a priority scheduler, the significance of its results in these simulations are limited. Due to time constraints, our knowledge of this scheduler is limited to the general idea. We did not have time to look into the implementation in the LTE model of NS-3 [3]. Also, the bandwidth of the congested link was defined by the LTE model. We saw that this bandwidth was 17.53 megabits per second.

In the following passages, we only use examples to visualize our findings. The full data and all plots can be found in Appendices A.2 and B.2.

4.3.1 Single Queue AQM

The the Single Queue AQM implementation in some cases achieved the target delay. The average delays were between 5.1 and 8.5 milliseconds for PIE and between 5.4 and 9.8 milliseconds for RED. The maximum delays were 18 and 46 for PIE and between 11 and 34 for RED.

Compared to the static link simulations, PIE achieved similar delays in both networks. RED achieved lower delays in the LTE network than in the static link network. We think the reason for this is that due to the higher bandwidth, the target delay is easier to keep since the serialization time of a packet decreases.

But due to these low delays, the link was often underutilized due to the queue running empty. Here, the losses were larger than in the static link network. PIE suffered losses up to 34% and RED up to 64%.

Like in the static link network, the bandwidth was shared fairly in most cases with the exception of ECN unfairness in RED (See Section 2.1).

Influence of Flow Counts

In the simulations with PIE, the flow counts did not show any clear influence on the queue delays. In the ones with RED, the delays tended to increase with the number of flows.

The utilization tended to increase with the flow counts, especially when increasing from two to six flows.

The Figures 4.15 and 4.16 show the resulting metrics from simulations with Single Queue AQM across all flow count combinations. These plots should visualize the influence of the flow counts. We chose the results from the simulations with Cubic as examples. The results from the simulations with Reno showed similar effects.
Figure 4.15: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the Single Queue AQM simulations using Cubic on all flows and activating ECN on the classic flows.

Figure 4.16: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the Single Queue AQM simulations using Cubic on all flows and deactivating ECN on the classic flows.
Influence of TCP Settings

Like in the static link network, deactivating ECN on the classic link lead to ECN unfairness. Activating ECN also lead to an increase in maximum queue delay. Like in the static link network, we think this is because unlike dropped packets, marked packets were enqueued anyway, adding to the queue size and delay.

Figure 4.17 shows the resulting metrics from simulations with Single Queue AQM across all available TCP settings. These plots should visualize the influence of the TCP settings. We chose the results from the simulations with 5 low-latency and 5 classic flows as an example. The results from the simulations with other flow configurations showed similar effects.

4.3.2 Dual Queue Uncoupled AQM

The results from these simulations differed from the simulations in the static link network. We assume, the reason for this is that here a the proportional fair scheduler was used instead of a weighted round robin.

Here, Dual Queue Uncoupled PIE very often achieved the target delay in the low-latency queue, independent from the use of classic or scalable congestion control on the low-latency flows. With classic congestion control, PIE achieved an average low-latency queue delays between 5.0 and 6.6 milliseconds and with scalable congestion control between 4.8 and 5.9 milliseconds. The main difference between scalable and classic congestion control for the PIE implementation was in the maximum low-latency queue delay. It decreased from between 23 and 50 milliseconds to between 8 and 34 milliseconds when scalable congestion control was used.

With the RED implementations, the target delay was only achieved in a few specific flow combinations. Furthermore, the average low-latency queue delay tended to be larger for scalable congestion control having been between 1.2 and 19 milliseconds for classic and
between 1.3 and 30 milliseconds for scalable congestion control. The use of scalable or classic congestion control had no significant impact on the maximum low-latency queue delay.

Like in the static link network simulations, the classic queue delays were in the order of hundreds of milliseconds. But again, this comes by design.

Compared to the static link network simulations, Dual Queue Uncoupled AQM tended to achieve lower delays. We think, there are two reasons for this. First, as in the Single Queue AQM case, the total bandwidth was higher so the transmission time per packet went down. Second, it seems like the scheduler assigned disproportionally much throughput to one traffic class if it had less flows than the other. We think, this lead to shorter dequeuing intervals.

As in the static link model, Dual Queue Uncoupled AQM achieved almost always full link utilization. The fairness depended on the flow counts.

Influence of Flow Counts
The main observation concerning flow counts was, that the proportional fair scheduler (See 2.5.2) assigned disproportionally much throughput to one traffic class if it has less flows than the other.
This lead to a number of other effects.

First, the low-latency queue delays in RED depended on the ratio between the low-latency and the classic flow count. If there were more classic than low-latency flows, the delays were lower. And if there were more low-latency than classic flows, the delays were higher.
We think, the reason for this are the RED parameters and the scheduler. The thresholds of RED were set using base values that were scaled according to the expected throughput (See Section 3.1.2). And the expected throughput was calculated assuming a fair distribution.
In case of more classic than low-latency flows, the scheduler assigned more throughput to the low-latency class than expected. Therefore, the thresholds were set too low, leading to a low delay. In case of more low-latency than classic flows, the scheduler assigned less throughput to the low-latency class than expected. In this case, the thresholds were set too high, leading to high delays.
This is an implementation error, raising the question how significant the results of the simulations with Dual Queue Uncoupled RED are.

Of course, this disproportionate assignment of throughput also lead to unfairness in favour of the class with less flows.
And as in the static link network simulations, Dual Queue Uncoupled AQM achieved almost full link utilization.

The low-latency queue delays of the PIE implementation were not affected by the disproportionate bandwidth distribution, since the PIE takes the target delay itself as a parameter.

Figure 4.18 shows the resulting metrics from simulations with Dual Queue Uncoupled AQM across all flow count combinations when using classic congestion controls on the low-latency flows. Figure 4.19 shows the same for when using scalable congestion controls on the low-latency flows. These plots should visualize the influence of the flow counts. We chose the results from the simulations with Cubic, DCTCP and ECN activated on classic flows as examples. The results from the simulations with other congestion controls or ECN deactivated on classic flows showed similar effects.

Influence of TCP Settings
In the case where scalable congestion control was used on the low-latency link, we saw some differences between Relentless and DCTCP.
Figure 4.18: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the Dual Queue Uncoupled AQM simulations using Cubic on all flows and activating ECN on the classic flows.

Figure 4.19: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the Dual Queue Uncoupled AQM simulations using DCTCP on low-latency flows, Cubic on classic flows and activating ECN on the classic flows.
For PIE, Relentless tended to lead to higher maximum low-latency queue delays than DCTCP. But we saw no difference in the average low-latency queue delay or the throughput between DCTCP and Relentless.

For RED, the low-latency class tended to achieve a lower average delay with Relentless than with DCTCP. At the same time, Relentless often also caused the low-latency class to give up more throughput.

We assume, the reasons for this are the same as in the static link model.

Figure 4.20 shows the resulting metrics from simulations with Dual Queue Uncoupled AQM across all available TCP settings when using classic congestion control on the low-latency flows. Figure 4.21 shows the same for when using scalable congestion control on the low-latency flows. These plots should visualize the influence of the TCP settings. We chose the results from the simulations with 5 low-latency and 5 classic flows as examples. The results from the simulations with other flow configurations showed similar effects.

### 4.3.3 DualQ Coupled AQM

In the simulations with the DualQ Coupled AQM, we saw that DualQ Coupled AQM definitely requires priority scheduling.

DualQ Coupled PIE achieved very low average L4S queue delays between 1.1 and 3.5 milliseconds and maximum delays between 4 and 20 milliseconds.

But this often lead to underutilization of the link with losses going up to 25%. Since the L4S queue had to give up throughput in order to achieve the low delay, the classic queue would have had to absorb it. Since the classic queue is also controlled by PIE, this was not always possible. The utilization mainly increased for higher classic flow counts. We think, this is because then the classic queue would run empty less often.
CHAPTER 4. RESULTS

Figure 4.21: Average and maximum queue delays, link utilization and modified Jain index over all available TCP settings for the Dual Queue Uncoupled AQM simulations with 5 low-latency and 5 classic flows when using scalable congestion control on low-latency flows and classic congestion control on low-latency flows.

Also here, the scheduler assigned disproportionally much throughput to the traffic class with fewer flows. Since the L4S queue was at a fundamental disadvantage, the classic queue always achieved a higher per-flow-throughput.

Figure 4.22 shows resulting metrics from simulations with DualQ Coupled AQM across all flow count combinations. These plots should visualize the influence of the flow counts. We chose the results from the simulations with DCTCP, Cubic and ECN activated on the classic flows as examples. The results from the simulations with other congestion controls or deactivated ECN on the classic flows showed similar effects.

The classic queue achieved average delays between 9.7 and 20 milliseconds and maximum delays between 38 and 88 milliseconds.

Like in the static link network simulations, DCTCP achieved higher throughputs than Relentless without any significant change in the low-latency queue delays. We think the reason for this is the same as in the static link network.

Figure 4.23 shows the resulting metrics from simulations with DualQ Coupled AQM across all available TCP settings. These plots should visualize the influence of the TCP settings. We chose the results from the simulations with 5 low-latency and 5 classic flows as an example. The results from the simulations with other flow configurations showed similar effects.

4.3.4 Comparison

We saw that the Single Queue AQM still could not achieve the target delay for higher flow counts. And it lost utilization when low delays were achieved.
Figure 4.22: Average and maximum queue delays, link utilization and modified Jain index over all flow count combinations for the DualQ Coupled AQM simulations using DCTCP on L4S flows, Cubic on classic flows and activating ECN on the classic flows.

Figure 4.23: Average and maximum queue delays, link utilization and modified Jain index over all available TCP settings for the DualQ Coupled AQM simulations with 5 low-latency and 5 classic flows.
In these simulations, Dual Queue Uncoupled PIE achieved the target delay without trading away a high link utilization. The use of scalable instead of classic congestion control, lead to a significant decrease in maximum low-latency queue delay. The average low-latency delay was more or less unchanged.

Dual Queue Uncoupled RED only achieved the target delay for certain flow combinations because of an implementation error.

DualQ Coupled AQM achieved low L4S queue delays but had to give up a lot of throughput to do so.

4.4 Conclusion

4.4.1 AQM Schemes

In our simulations we saw that the Single Queue AQM implementations had limits concerning achievable delays and link utilization. Low target delays could only be achieved in cases with very few flows. For higher numbers of flows, the available queue size corresponding to the delay seemed to be too small for AQM schemes to work on.

Also, low delays came at the cost of link utilization. This is a known problem described in Section 2.2.3.

Compared to the Single Queue AQM, Dual Queue Uncoupled AQM implementations were able to achieve full link utilization while trying to achieve low delays in the low-latency queue. Concerning delays, we saw that the scheduling of the two queues could cause problems. In the static link network simulations, where we used a weighted round robin scheduler, this scheduler lead to increased delays when there were much more classic than low-latency flows. In the LTE network simulations, where a proportional fair scheduler had to be used, this problem was alleviated at the cost of fairness. Other schedulers might further improve the performance of this algorithm. Depending on the requirements, this could be seen as a better performance.

The Dual Queue Uncoupled AQM approach also allowed the use of scalable congestion control. While a high link utilization was given by design, using classic congestion controls often lead to the low-latency queue having to give up assigned throughput because it ran empty (See Section 2.2.3). Using an elaborate scalable congestion control algorithm like DCTCP allowed the low-latency queue to fully use the assigned throughput in most cases. The less elaborate Relentless algorithm on the other hand, in some cases even deteriorated the fairness further.

In order to achieve full link utilization, the classic queue was kept full at all times. In some cases, this lead to massive classic queue delays in the order of hundreds or thousands of milliseconds. We think, this something that should be taken into consideration.

The DualQ Coupled AQM implementations were able to achieve very low delays. In the LTE network simulations, we saw that the priority scheduling is a very important part of this algorithm. Here, we had to use a proportional fair scheduler which caused the L4S queue to give up a lot of throughput while trying to achieve the low delay.

DualQ Coupled AQM also lost utilization when the classic queue was running empty. Possibly, this could be solved by increasing the target delays used in the AQM schemes.

Unlike with Dual Queue Uncoupled AQM, the average classic queue delays with DualQ Coupled AQM stayed below 100 milliseconds and the maximum classic queue delays all stayed below 350 milliseconds. This might be an advantage over Dual Queue Uncoupled AQM.

4.4.2 TCP Settings

For Single Queue AQM and Dual Queue Uncoupled AQM, we ran simulations using the classic congestion controls Cubic and Reno. In these simulations we saw that in most cases, Cubic achieved lower maximum delays and higher throughput than Reno.
For Dual Queue Uncoupled and DualQ Coupled AQM, we ran simulations using the scalable congestion controls DCTCP and Relentless. We saw that Relentless often achieved lower lower delays than DCTCP. At the same time, DCTCP achieved higher throughput than Relentless.

The main influence of turning on ECN on the classic flows or not were the occurrences of ECN unfairness. Aside from that, ECN caused an increase in maximum queue delays for Single Queue AQM.
Chapter 5
Summary and Outlook

5.1 Summary

In this work, we wanted to evaluate the performance of AQM schemes that can be used in the internet. We wanted to find out which algorithms can achieve very low queuing delays in order to serve the low-latency requirement that an increasing amount of applications and traffic has today. Furthermore, we wanted to see what concessions have to be made in order to achieve these low delays.

In order to answer these questions, we implemented different AQM algorithms for the network simulator NS-3 and ran simulations. Based on the results from these simulations, we evaluated the performance of the algorithms. For the first simulations we were using a static link model, simulating a wired network. In the second, we used an LTE link model. This should give insight into how these algorithms perform in wired and in mobile networks.

We implemented three algorithms.
The first algorithm used a single queue, controlled by a traditional AQM scheme.
The second algorithm was the DualQ Coupled AQM algorithm from [26]. It uses two queues to separate traffic based on whether they seek low latency or not. The queue of the low-latency traffic is scheduled with priority. A traditional AQM schemes is used for controlling the two queues with a coupling between them.
The third algorithm was a simplification of DualQ Coupled AQM. It also separates the traffic into two queues. Both queues run independent instances of a traditional AQM but with different target delays. The two queues are scheduled by a weighted round robin scheduler.

In our static link network simulations, we saw that the algorithms using a single queue often failed to achieve a low delay and that they lost link utilization by trying.
The simplification of DualQ Coupled AQM also often failed to achieve the low target delay. But this implementation always achieved full link utilization due to its design.
DualQ Coupled AQM achieved very low delays. But this algorithm sometimes also lost link utilization.

Our LTE network simulations were hindered by implementation issues. The DualQ Coupled AQM algorithm could not be implemented with priority scheduling. We therefore had to use a proportional fair scheduler. For the simplification of DualQ Coupled AQM, we also had to use a proportional fair scheduler.
These simulations also showed that the algorithms with a single queue often failed to keep a low target delay. And that trying to achieve it lead to underutilization of the link.
Here, DualQ Coupled also achieved very low delays but the low latency seeking traffic only achieved little throughput. this is probably due to the lack of priority scheduling.
And the simplification of DualQ Coupled AQM was in most cases able to achieve the low target delay which came in some cases at the cost of fairness but without loss of link utilization.
5.2 Outlook

In the future, the DualQ Coupled AQM algorithm from [26] could be correctly implemented for the LTE module of NS-3. This could show whether the low delays achieved using the static link model can also be achieved in a mobile environment.

Also, Dual Queue Uncoupled AQM could be implemented with another scheduler than a weighted round robin. Since the simulations in the LTE network with a proportional fair scheduler resulted in lower delays, changing the scheduler could also lead to lower delays when using the static link model.
Appendix A

Plots

A.1 Plots from Static Link Simulations

[Graphs and charts showing data from static link simulations]
APPENDIX A. PLOTS

Single Queue AQM

Queue Delay

Utilization and Modified Jain Index

Single Queue AQM
Low-Latency Flows: 1 - Classic Flows: 10

Queue Delay

Utilization and Modified Jain Index
APPENDIX A. PLOTS

Single Queue AQM
Low-Latency Flows: 10 - Classic Flows: 1

Queue Delay

Utilization and Modified Jain Index

Single Queue AQM
Low-Latency Flows: 10 - Classic Flows: 1

Queue Delay

Utilization and Modified Jain Index
APPENDIX A. PLOTS

Single Queue AQM
Congestion Control: Cubic - ECN on Classic flows: On

Queue Delay

Utilization and Modified Jain Index

Single Queue AQM
Congestion Control: Cubic - ECN on Classic flows: Off

Queue Delay

Utilization and Modified Jain Index
APPENDIX A. PLOTS

Dual Queue Uncoupled AQM with Classic Congestion Control
Low-Latency Flows: 10 - Classic Flows: 10

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Classic Congestion Control on all Flows
Congestion Control: Cubic - ECN on Classic flows: On

Low-Latency Queue Delay

Utilization and Modified Jain Index
Dual Queue Uncoupled AQM with Classic Congestion Control on all Flows
Congestion Control: Cubic - ECN on Classic flows: Off

Low-Latency Queue Delay
- Maximum Queue Delay: RED
- Average Queue Delay: RED
- Maximum Queue Delay: PE
- Average Queue Delay: PE

Utilization and Modified Jain Index
- Link Utilization: RED
- Modified Jain Index: RED
- Link Utilization: PE
- Modified Jain Index: PE

Dual Queue Uncoupled AQM with Classic Congestion Control on all Flows
Congestion Control: Reno - ECN on Classic flows: On

Low-Latency Queue Delay
- Maximum Queue Delay: RED
- Average Queue Delay: RED
- Maximum Queue Delay: PE
- Average Queue Delay: PE

Utilization and Modified Jain Index
- Link Utilization: RED
- Modified Jain Index: RED
- Link Utilization: PE
- Modified Jain Index: PE
APPENDIX A. PLOTS

Dual Queue Uncoupled AQM with Classic Congestion Control on all Flows
Congestion Control: Reno - ECN on Classic flows: Off

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Scalable Congestion Control
Low-Latency Flows: 1 - Classic Flows: 1

Low-Latency Queue Delay

Utilization and Modified Jain Index
Dual Queue Uncoupled AQM with Scalable Congestion Control

Dual Queue Uncoupled AQM with Scalable Congestion Control
Low-Latency Flows: 1 - Classic Flows: 10
A.1 Plots from Static Link Simulations

Dual Queue Uncoupled AQM with Scalable Congestion Control
Low-Latency Flows: 10 - Classic Flows: 1
Dual Queue Uncoupled AQM with Scalable Congestion Control
Low-Latency Flows: 10 - Classic Flows: 5

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Scalable Congestion Control
Low-Latency Flows: 10 - Classic Flows: 10

Low-Latency Queue Delay

Utilization and Modified Jain Index
A.1 Plots from Static Link Simulations

Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows
Classic CC: Cubic - Scalable CC: DCTCP - ECN on Classic flows: On

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows
Classic CC: Cubic - Scalable CC: DCTCP - ECN on Classic flows: Off

Low-Latency Queue Delay

Utilization and Modified Jain Index
A.1 Plots from Static Link Simulations

Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows
Classic CC: Reno - Scalable CC: DCTCP - ECN on Classic flows: On

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows
Classic CC: Reno - Scalable CC: DCTCP - ECN on Classic flows: Off

Low-Latency Queue Delay

Utilization and Modified Jain Index
Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows
Classic CC: Reno - Scalable CC: Relentless - ECN on Classic flows: Off

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows
Classic CC: Reno - Scalable CC: Relentless - ECN on Classic flows: On

Low-Latency Queue Delay

Utilization and Modified Jain Index
A.1 Plots from Static Link Simulations

DualQ Coupled AQM
L4S Flows: 5 - Classic Flows: 1

DualQ Coupled AQM
L4S Flows: 10 - Classic Flows: 1
APPENDIX A. PLOTS

DualQ Coupled AQM
L4S Flows: 10 - Classic Flows: 1

L4S Queue Delay

Utilization and Modified Jain Index

DualQ Coupled AQM
L4S Flows: 10 - Classic Flows: 5

L4S Queue Delay

Utilization and Modified Jain Index
A.1 Plots from Static Link Simulations

DualQ Coupled AQM
L4S Flows: 10 - Classic Flows: 10

DualQ Coupled AQM
Classic CC: Cubic - Scalable CC: DCTCP - ECN on Classic flows: On
Appendix A. Plots

DualQ Coupled AQM
Classic CC: Reno - Scalable CC: Relentless - ECN on Classic flows: On

DualQ Coupled AQM
Classic CC: Reno - Scalable CC: DCTCP - ECN on Classic flows: Off
A.2 Plots from LTE Link Simulations

DualQ Coupled AQM
Classic CC: Reno - Scalable CC: Relentless - ECN on Classic flows: Off

L4S Queue Delay

Utilization and Modified Jain Index

Single Queue AQM (LTE)
Low-Latency Flows: 1 - Classic Flows: 1

Queue Delay

Utilization and Modified Jain Index
APPENDIX A. PLOTS

Single Queue AQM (LTE)

Queue Delay

Utilization and Modified Jain Index

Single Queue AQM (LTE)
Low-Latency Flows: 1 - Classic Flows: 10

Queue Delay

Utilization and Modified Jain Index
Single Queue AQM (LTE)  
Congestion Control: Cubic - ECN on Classic flows: On

Single Queue AQM (LTE)  
Congestion Control: Cubic - ECN on Classic flows: Off
Dual Queue Uncoupled AQM with Classic Congestion Control (LTE)
Low-Latency Flows: 5 - Classic Flows: 1

Dual Queue Uncoupled AQM with Classic Congestion Control (LTE)
Low-Latency Flows: 10 - Classic Flows: 1
Dual Queue Uncoupled AQM with Classic Congestion Control (LTE)
Low-Latency Flows: 10 - Classic Flows: 1

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Classic Congestion Control (LTE)
Low-Latency Flows: 10 - Classic Flows: 5

Low-Latency Queue Delay

Utilization and Modified Jain Index
Dual Queue Uncoupled AQM with Classic Congestion Control (LTE)
Low-Latency Flows: 10 - Classic Flows: 10

Dual Queue Uncoupled AQM with Classic Congestion Control on all Flows (LTE)
Congestion Control: Cubic - ECN on Classic flows: On
Dual Queue Uncoupled AQM with Classic Congestion Control on all Flows (LTE)
Congestion Control: Reno - ECN on Classic flows: Off

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Scalable Congestion Control (LTE)
Low-Latency Flows: 1 - Classic Flows: 1

Low-Latency Queue Delay

Utilization and Modified Jain Index
A.2 Plots from LTE Link Simulations

Dual Queue Uncoupled AQM with Scalable Congestion Control (LTE)

Dual Queue Uncoupled AQM with Scalable Congestion Control (LTE)
Low-Latency Flows: 1 - Classic Flows: 10
Dual Queue Uncoupled AQM with Scalable Congestion Control (LTE)
Low-Latency Flows: 5 - Classic Flows: 1

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Scalable Congestion Control (LTE)
Low-Latency Flows: 5 - Classic Flows: 1

Low-Latency Queue Delay

Utilization and Modified Jain Index
A.2 Plots from LTE Link Simulations

Dual Queue Uncoupled AQM with Scalable Congestion Control (LTE)

Low-Latency Flows: 10 - Classic Flows: 1

Low-Latency Queue Delay

Utilization and Modified Jain Index (L)

Dual Queue Uncoupled AQM with Scalable Congestion Control (LTE)

Low-Latency Flows: 10 - Classic Flows: 1

Low-Latency Queue Delay

Utilization and Modified Jain Index (L)
Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows (LTE)
Classic CC: Cubic - Scalable CC: Relentless - ECN on Classic flows: On

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows (LTE)
Classic CC: Cubic - Scalable CC: Relentless - ECN on Classic flows: Off

Low-Latency Queue Delay

Utilization and Modified Jain Index
Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows (LTE)
Classic CC: Reno - Scalable CC: DCTCP - ECN on Classic flows: On

Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows (LTE)
Classic CC: Reno - Scalable CC: DCTCP - ECN on Classic flows: Off
Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows (LTE)
Classic CC: Reno - Scalable CC: Relentless - ECN on Classic flows: On

Low-Latency Queue Delay

Utilization and Modified Jain Index

Dual Queue Uncoupled AQM with Scalable Congestion Control on Low-Latency Flows (LTE)
Classic CC: Reno - Scalable CC: Relentless - ECN on Classic flows: Off

Low-Latency Queue Delay

Utilization and Modified Jain Index
A.2 Plots from LTE Link Simulations

DualQ Coupled AQM (LTE)
L4S Flows: 1 - Classic Flows: 1

DualQ Coupled AQM (LTE)
APPENDIX A. PLOTS

Dual AQM Coupled (LTE)
L4S Flows: 1 - Classic Flows: 10

Dual AQM Coupled (LTE)
L4S Flows: 5 - Classic Flows: 1
A.2 Plots from LTE Link Simulations

DualQ Coupled AQM (LTE)
L4S Flows: 5 - Classic Flows: 1

L4S Queue Delay

- Maximum Queue Delay - PE
- Average Queue Delay - PE

Utilization and Modified Jain Index

- Link Utilization - PE
- Modified Jain Index - PE

DualQ Coupled AQM (LTE)
L4S Flows: 10 - Classic Flows: 1

L4S Queue Delay

- Maximum Queue Delay - PE
- Average Queue Delay - PE

Utilization and Modified Jain Index

- Link Utilization - PE
- Modified Jain Index - PE
A.2 Plots from LTE Link Simulations

**DualQ Coupled AQM (LTE)**
Classic CC: Cubic - Scalable CC: Relentless - ECN on Classic flows: Off

![Queue Delay Graph](image1)

![Utilization and Modified Jain Index](image2)

**DualQ Coupled AQM (LTE)**
Classic CC: Reno - Scalable CC: DCTCP - ECN on Classic flows: On

![Queue Delay Graph](image3)

![Utilization and Modified Jain Index](image4)
DualQ Coupled AQM (LTE)  
Classic CC: Reno - Scalable CC: Relentless - ECN on Classic flows: On

DualQ Coupled AQM (LTE)  
Classic CC: Reno - Scalable CC: DCTCP - ECN on Classic flows: Off
Appendix B

Data

This appendix contains the data from our simulations. Each line describes the set-up and metrics from one simulation.

*aqm* denotes the AQM scheme that was used.

- *sq – pie* stands for Single Queue AQM with PIE.
- *sq – red* stands for Single Queue AQM with RED.
- *dq – pie* stands for Dual Queue Uncoupled AQM with PIE.
- *dq – red* stands for Dual Queue Uncoupled AQM with RED.
- *l4s – cred* stands for DualQ Coupled AQM with PIE.
- *l4s – pi2* stands for DualQ Coupled AQM with RED.
- *l4s – pie* stands for DualQ Coupled AQM with PIE.

*nLF* and *nCF* denote, how many low-latency/L4S flows and how many classic flows were simulated.

*ccC* and *ccL* denote the congestion controls that were used for the classic and the low-latency/L4S flows.

*utilization* gives the achieved link utilization.

*jainmod* gives the value of the modified Jain index (See Section 4.1).

*qDelayAvgC* and *qDelayAvgL* give the average queue delays of the classic and the low-latency/L4S queue in milliseconds.

*qDelayMaxC* and *qDelayMaxL* give the maximum queue delays of the classic and the low-latency/L4S queue in milliseconds.
### B.1 Data from Static Link Simulations

| sq-pie | 1 | 5 | cubic | cubic | 0 | 0.9844708533 | 1.0003243656 | 6.5527367117 | 26.1002000000 | 6.5527367117 | 26.1002000000 |
| sq-pie | 1 | 5 | cubic | cubic | 1 | 0.9665670233 | 1.0018909343 | 32.1042000000 | 32.1042000000 | 32.1042000000 | 32.1042000000 |
| sq-pie | 1 | 10 | reno | reno | 0 | 0.9263334333 | 1.0014541659 | 8.2447015062 | 38.1162000000 | 8.2447015062 | 38.1162000000 |
| sq-pie | 1 | 10 | reno | reno | 1 | 0.8991721000 | 1.0007212000 | 3.7736630839 | 40.5194000000 | 3.7736630839 | 40.5194000000 |
| sq-pie | 10 | 5 | cubic | cubic | 0 | 0.9902060700 | 1.0007022832 | 6.2246181596 | 26.1122000000 | 6.2246181596 | 26.1122000000 |
| sq-pie | 10 | 5 | reno | reno | 0 | 0.9374622000 | 1.0000397200 | 6.8314430767 | 33.3098000000 | 6.8314430767 | 33.3098000000 |
| sq-pie | 10 | 5 | reno | reno | 1 | 0.9208909343 | 0.9902058800 | 6.5527367117 | 26.1002000000 | 6.5527367117 | 26.1002000000 |
| sq-pie | 10 | 10 | reno | reno | 0 | 0.9831921333 | 1.0009398135 | 6.0548691712 | 33.3098000000 | 6.0548691712 | 33.3098000000 |
| sq-red | 1 | 1 | cubic | cubic | 1 | 0.9317665667 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 1 | 1 | cubic | cubic | 1 | 0.9905526777 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 1 | 5 | cubic | cubic | 0 | 0.9195451000 | 0.9905526777 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 1 | 5 | cubic | cubic | 0 | 0.9195451000 | 0.9905526777 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 1 | 10 | cubic | cubic | 0 | 0.9495593000 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 1 | 10 | cubic | cubic | 0 | 0.9495593000 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 5 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 5 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 10 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 10 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 10 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 10 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 10 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 10 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 10 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 10 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 10 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
| sq-red | 10 | 10 | cubic | cubic | 0 | 0.9664464333 | 0.9999902000 | 0.9844708533 | 1.0003243656 | 0.9844708533 | 1.0003243656 |
### B.1 Data from Static Link Simulations

<table>
<thead>
<tr>
<th>Model</th>
<th>Num</th>
<th>Den</th>
<th>Method</th>
<th>Method</th>
<th>Loss</th>
<th>Latency</th>
<th>Throughput</th>
<th>Load</th>
<th>Throughput</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>sq-red</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>reno</td>
<td>0</td>
<td>0.9924841933</td>
<td>1.0543859685</td>
<td>19.8754749348</td>
<td>44.1362000000</td>
<td>19.8754749348</td>
</tr>
<tr>
<td>sq-red</td>
<td>5</td>
<td>1</td>
<td>reno</td>
<td>reno</td>
<td>0</td>
<td>0.9750609900</td>
<td>0.9996545377</td>
<td>18.9414346195</td>
<td>47.7290000000</td>
<td>18.9414346195</td>
</tr>
<tr>
<td>sq-red</td>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>cubic</td>
<td>0</td>
<td>0.9997062333</td>
<td>1.1493913803</td>
<td>23.7773029182</td>
<td>44.1242000000</td>
<td>23.7773029182</td>
</tr>
<tr>
<td>sq-red</td>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>cubic</td>
<td>0</td>
<td>0.9993588000</td>
<td>1.3780859882</td>
<td>23.6034544850</td>
<td>45.3258000000</td>
<td>23.6034544850</td>
</tr>
<tr>
<td>sq-red</td>
<td>5</td>
<td>10</td>
<td>reno</td>
<td>reno</td>
<td>0</td>
<td>0.9871272500</td>
<td>1.4223637062</td>
<td>21.2670814879</td>
<td>48.0399000000</td>
<td>21.2670814879</td>
</tr>
<tr>
<td>sq-red</td>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>cubic</td>
<td>1</td>
<td>0.9862433867</td>
<td>0.9999945517</td>
<td>24.9203072121</td>
<td>52.5354000000</td>
<td>24.9203072121</td>
</tr>
<tr>
<td>sq-red</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>reno</td>
<td>1</td>
<td>0.8993877200</td>
<td>1.0006675565</td>
<td>19.9235235206</td>
<td>60.0559000000</td>
<td>19.9235235206</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>1</td>
<td>cubic</td>
<td>cubic</td>
<td>0</td>
<td>0.9999942333</td>
<td>0.9937148971</td>
<td>5.7202012670</td>
<td>21.2938000000</td>
<td>368.5985881270</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>1</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9999921000</td>
<td>0.9999999996</td>
<td>6.2837088985</td>
<td>11.6810000000</td>
<td>251.3012560260</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>1</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9994089167</td>
<td>1.0000001043</td>
<td>6.7848383888</td>
<td>16.9554000000</td>
<td>380.0180933280</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>1</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9999922000</td>
<td>0.9999999996</td>
<td>6.2250776361</td>
<td>11.6810000000</td>
<td>379.9992854810</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>1.0000000000</td>
<td>0.999992925</td>
<td>6.6627857116</td>
<td>15.6088000000</td>
<td>335.8530158590</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>1</td>
<td>reno</td>
<td>reno</td>
<td>1</td>
<td>0.9999915333</td>
<td>0.9704120621</td>
<td>6.2795455952</td>
<td>26.1002000000</td>
<td>299.0231573010</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>5</td>
<td>cubic</td>
<td>cubic</td>
<td>0</td>
<td>0.9998391167</td>
<td>0.9994362418</td>
<td>13.1310851815</td>
<td>28.5034000000</td>
<td>410.0419165460</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9998887333</td>
<td>1.0000002175</td>
<td>20.0236253200</td>
<td>42.0546000000</td>
<td>406.2486659030</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>1.0000000000</td>
<td>0.9999102877</td>
<td>13.6431227446</td>
<td>42.0319000000</td>
<td>408.3491262340</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>5</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>1.0000000000</td>
<td>0.9999992925</td>
<td>20.0846149345</td>
<td>35.7010000000</td>
<td>335.9319092110</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>1.0000000000</td>
<td>0.9999059385</td>
<td>13.8091273199</td>
<td>35.7130000000</td>
<td>339.6516969040</td>
</tr>
<tr>
<td>dq-pie</td>
<td>1</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>1.0000000000</td>
<td>0.9998852792</td>
<td>13.9414250859</td>
<td>41.7210000000</td>
<td>371.6242861520</td>
</tr>
<tr>
<td>dq-pie</td>
<td></td>
<td></td>
<td></td>
<td>dctcp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>---</td>
<td>---</td>
<td>------</td>
<td>-------</td>
<td>---</td>
<td>------------</td>
<td>----------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>0</td>
<td>0.9992010433</td>
<td>1.0000011267</td>
<td>28.5664192485</td>
<td>69.7033000000</td>
<td>407.9049652060</td>
<td>720.6240000000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>1</td>
<td>0.9999716400</td>
<td>0.9999999996</td>
<td>28.7512988546</td>
<td>57.3298000000</td>
<td>412.3903183610</td>
<td>1078.6900000000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>0</td>
<td>0.9999916833</td>
<td>1.0000000020</td>
<td>28.8474809209</td>
<td>57.3298000000</td>
<td>351.9870329130</td>
<td>540.3850000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>cubic</td>
<td>0</td>
<td>0.9999990833</td>
<td>0.9999031474</td>
<td>6.9169892589</td>
<td>20.0922000000</td>
<td>416.1359306680</td>
<td>709.7980000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>cubic</td>
<td>1</td>
<td>1.0000000000</td>
<td>0.9995617371</td>
<td>6.2399108988</td>
<td>21.2818000000</td>
<td>363.2438997470</td>
<td>1085.0300000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>reno</td>
<td>0</td>
<td>1.0000000000</td>
<td>0.9996824535</td>
<td>6.1714308795</td>
<td>20.0922000000</td>
<td>333.6137916690</td>
<td>547.5940000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>0</td>
<td>0.9996313500</td>
<td>0.9977487320</td>
<td>7.4141487803</td>
<td>30.0280000000</td>
<td>419.7185488800</td>
<td>858.7960000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>1</td>
<td>1.0000000000</td>
<td>0.9998922436</td>
<td>7.8405541636</td>
<td>27.3018000000</td>
<td>396.3873714960</td>
<td>1102.3000000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>relentless</td>
<td>1</td>
<td>0.9999941333</td>
<td>0.9992812855</td>
<td>7.7603668992</td>
<td>28.5034000000</td>
<td>386.1948222470</td>
<td>1135.4900000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>relentless</td>
<td>0</td>
<td>0.9999915333</td>
<td>0.9993069332</td>
<td>7.9107784196</td>
<td>30.9066000000</td>
<td>352.1763329690</td>
<td>545.5140000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>relentless</td>
<td>1</td>
<td>0.9999915833</td>
<td>0.9992201112</td>
<td>7.9287352364</td>
<td>27.6127000000</td>
<td>356.2469596100</td>
<td>569.2230000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>reno</td>
<td>1</td>
<td>1.0000000000</td>
<td>0.9891124007</td>
<td>8.9245239635</td>
<td>45.3258000000</td>
<td>345.0681240600</td>
<td>581.2390000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>0</td>
<td>0.9988226667</td>
<td>0.9970912016</td>
<td>9.3207413998</td>
<td>35.7130000000</td>
<td>410.8963992750</td>
<td>803.9020000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>1</td>
<td>0.9999941833</td>
<td>0.9966122743</td>
<td>9.2101251519</td>
<td>35.7130000000</td>
<td>402.9550246380</td>
<td>1157.2600000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>0</td>
<td>0.9994586667</td>
<td>0.9997246150</td>
<td>10.6633713183</td>
<td>37.2537000000</td>
<td>411.9472688320</td>
<td>742.4570000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>reno</td>
<td>0</td>
<td>1.0000000000</td>
<td>0.9996575643</td>
<td>10.5390716172</td>
<td>35.7130000000</td>
<td>369.3463995670</td>
<td>561.8440000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>reno</td>
<td>1</td>
<td>1.0000000000</td>
<td>0.9996653754</td>
<td>10.4834855941</td>
<td>36.9146000000</td>
<td>358.3294328470</td>
<td>564.4170000000</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>1</td>
<td>0.9999313767</td>
<td>0.9992474661</td>
<td>6.0062352608</td>
<td>20.1043000000</td>
<td>455.4394139110</td>
<td>1333.4500000000</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>0</td>
<td>0.9999265500</td>
<td>0.9944872872</td>
<td>6.1342913233</td>
<td>28.5034000000</td>
<td>392.1776903750</td>
<td>861.2120000000</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>relentless</td>
<td>0</td>
<td>0.9999265500</td>
<td>0.9944872872</td>
<td>6.1245732889</td>
<td>26.1002000000</td>
<td>394.5124634570</td>
<td>1220.4900000000</td>
<td></td>
</tr>
</tbody>
</table>
### B.1 Data from Static Link Simulations

| dq-pie | 10 | 1 | reno | dctcp | 0 | 0.9999715867 | 0.9992444025 | 6.0220036642 | 20.0922000000 | 359.4297259360 | 658.1410000000 |
| dq-pie | 10 | 1 | reno | dctcp | 1 | 1.0000000000 | 0.9993377268 | 6.0529612983 | 22.8184000000 | 368.4480198090 | 665.3510000000 |
| dq-pie | 10 | 5 | cubic | dctcp | 0 | 0.9999692000 | 0.9993246098 | 7.3608390676 | 28.8264000000 | 414.4329469520 | 777.8880000000 |
| dq-pie | 10 | 5 | cubic | dctcp | 1 | 1.0000000000 | 0.9994148374 | 7.2991281831 | 30.0159000000 | 427.2296945860 | 1174.8200000000 |
| dq-pie | 10 | 5 | cubic | relentless | 0 | 0.9997164000 | 0.9969644006 | 7.5388396848 | 28.5591000000 | 417.4482306310 | 808.3290000000 |
| dq-pie | 10 | 5 | reno | dctcp | 0 | 0.9999916000 | 0.9994765802 | 7.3825038262 | 24.8986000000 | 371.8678035620 | 563.2150000000 |
| dq-pie | 10 | 5 | reno | dctcp | 1 | 0.9998113500 | 0.9995145330 | 7.2573390386 | 26.1002000000 | 359.2971746350 | 584.8440000000 |
| dq-pie | 10 | 5 | reno | reno | 0 | 0.9999914833 | 0.9870167990 | 8.1974864148 | 38.1042000000 | 385.9478639180 | 589.3410000000 |
| dq-pie | 10 | 5 | reno | reno | 1 | 0.9992305333 | 0.9855470189 | 8.4229295176 | 49.2415000000 | 351.0634049140 | 617.6100000000 |
| dq-pie | 10 | 10 | cubic | dctcp | 1 | 0.9999915500 | 0.9994762898 | 9.7131541971 | 38.1162000000 | 470.6813658220 | 1314.2100000000 |
| dq-pie | 10 | 10 | reno | relentless | 0 | 0.9999914667 | 0.9971126091 | 9.9433067000 | 42.3966000000 | 370.5745452270 | 588.4490000000 |
| dq-red | 1 | 1 | cubic | dctcp | 0 | 0.9999919500 | 0.9999974969 | 6.0258114599 | 14.0722000000 | 224.0437515900 | 416.9390000000 |
| dq-red | 1 | 1 | cubic | relentless | 0 | 0.9999790000 | 0.9945556500 | 3.8335587199 | 14.0722000000 | 209.7181512440 | 357.7440000000 |
| dq-red | 1 | 1 | reno | dctcp | 0 | 0.9999919333 | 0.9999976219 | 6.0645481537 | 14.0722000000 | 201.4886551080 | 300.0670000000 |
| dq-red | 1 | 1 | reno | reno | 0 | 0.9992506667 | 0.9853595022 | 8.3080171263 | 39.6408000000 | 153.6242470460 | 299.1860000000 |
| dq-red | 1 | 5 | cubic | cubic | 1 | 0.9999966333 | 0.9998650218 | 24.4221270012 | 57.3418000000 | 180.2807166050 | 263.1260000000 |
| dq-red | 1 | 5 | cubic | relentless | 0 | 1.0000000000 | 0.9988830768 | 6.4981995499 | 28.4914000000 | 182.4957047500 | 277.2460000000 |
| dq-red | 1 | 5 | reno | dctcp | 0 | 1.0000000000 | 0.9999998839 | 11.1955833658 | 21.2938000000 | 155.3732707460 | 213.5500000000 |
| dq-red | 1 | 5 | reno | dctcp | 1 | 0.9999912000 | 0.9999999863 | 11.2855084978 | 28.4914000000 | 158.3061691470 | 221.9610000000 |
| dq-red | 1 | 5 | reno | relentless | 1 | 0.9999915500 | 0.9986658453 | 6.4829203475 | 28.4914000000 | 157.9794272920 | 217.1540000000 |
| dq-red | 1 | 5 | reno | reno | 0 | 1.0000000000 | 0.9976440099 | 19.8511654138 | 57.3538000000 | 153.6632574880 | 215.9550000000 |
| dq-red | 1 | 10 | reno | relentless | 0 | 1.0000000000 | 0.9992944554 | 12.2965601127 | 30.9186000000 | 176.8407871020 | 238.7830000000 |
| dq-red | 1 | 10 | reno | reno | 1 | 0.9999990767 | 0.9959675533 | 21.5112218220 | 57.3538000000 | 178.9724293580 | 239.9850000000 |
| dq-red | 5 | 1 | cubic | cubic | 0 | 0.9999938500 | 0.9999817220 | 20.7949637737 | 44.1242000000 | 899.2484681810 | 1935.4400000000 |
| dq-red | 5 | 1 | cubic | dctcp | 0 | 1.0000000000 | 0.9987722152 | 5.2578111809 | 20.0922000000 | 800.3399412290 | 1825.3300000000 |
| dq-red | 5 | 1 | cubic | relentless | 0 | 0.9996563333 | 0.9075071168 | 3.7338559010 | 18.8786000000 | 427.2165224120 | 977.7550000000 |
| dq-red | 5 | 1 | reno | dctcp | 1 | 0.9999991833 | 0.9987293145 | 5.3027140581 | 18.8906000000 | 603.8972448530 | 825.1640000000 |
| dq-red | 5 | 1 | reno | relentless | 0 | 1.0000000000 | 0.8923248509 | 3.6987167243 | 16.8104000000 | 321.1232028730 | 715.8180000000 |
| dq-red | 5 | 1 | reno | reno | 1 | 0.9999940000 | 0.9889860523 | 18.1836505962 | 50.1322000000 | 502.8452162480 | 835.9780000000 |
| dq-red | 5 | 5 | cubic | cubic | 1 | 0.9999970167 | 0.9997231117 | 23.5459639814 | 54.9386000000 | 340.6224529770 | 477.8890000000 |
| dq-red | 5 | 5 | cubic | relentless | 0 | 0.9999947833 | 0.9777159930 | 5.2965909201 | 30.0280000000 | 290.0051727880 | 461.4120000000 |
| dq-red | 5 | 5 | cubic | relentless | 1 | 1.0000000000 | 0.9802260412 | 5.2216288642 | 26.0882000000 | 288.3745993400 | 427.7570000000 |
| dq-red | 5 | 5 | reno | relentless | 0 | 1.0000000000 | 0.9792259426 | 5.1917917532 | 24.0079000000 | 231.9401105830 | 352.0560000000 |
| dq-red | 5 | 5 | reno | relentless | 1 | 1.0000000000 | 0.9787464978 | 5.1632876249 | 22.1777000000 | 232.2964373840 | 358.9430000000 |
| dq-red | 5 | 5 | reno | reno | 0 | 0.9999791500 | 0.9885309331 | 19.9432421411 | 61.3411000000 | 245.0704996240 | 343.3220000000 |
| dq-red | 5 | 10 | cubic | dctcp | 0 | 1.0000000000 | 0.9992646121 | 9.1465232750 | 35.7489000000 | 312.9780895430 | 394.2300000000 |
| dq-red | 5 | 10 | reno | dctcp | 0 | 1.0000000000 | 0.9991219159 | 9.2182655441 | 33.3098000000 | 242.9381114180 | 305.8370000000 |
| dq-red | 5 | 10 | reno | dctcp | 1 | 0.9999990833 | 0.9992133554 | 9.1176227218 | 35.7130000000 | 254.9773042440 | 330.4280000000 |
| dq-red | 5 | 10 | reno | reno | 1 | 0.9999865500 | 0.9887172011 | 22.4337025379 | 75.3658000000 | 239.7990337670 | 352.9350000000 |
| dq-red | 10 | 1 | cubic | cubic | 1 | 1.0000000000 | 0.9979123182 | 24.6817602105 | 51.3458000000 | 1481.7685414400 | 2233.4300000000 |
### B.1 Data from Static Link Simulations

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Window Size</th>
<th>Cwnd</th>
<th>Size</th>
<th>RTT</th>
<th></th>
<th>Delay</th>
<th>Throughput</th>
<th>Throughput</th>
<th>Throughput</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td></td>
<td></td>
<td></td>
<td>0.9999892</td>
<td>0.9895989</td>
<td>6.6225686</td>
<td>26.06900</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>relentless</td>
<td></td>
<td></td>
<td></td>
<td>0.9995187</td>
<td>0.7863008</td>
<td>4.8246947</td>
<td>29.69300</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>relentless</td>
<td></td>
<td></td>
<td></td>
<td>1.0000000</td>
<td>0.7532243</td>
<td>4.8229848</td>
<td>24.88660</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>dctcp</td>
<td></td>
<td></td>
<td>1.0000000</td>
<td>0.9892481</td>
<td>6.4837823</td>
<td>23.69700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>relentless</td>
<td></td>
<td></td>
<td></td>
<td>1.0000000</td>
<td>0.7722083</td>
<td>4.8729548</td>
<td>22.50750</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>relentless</td>
<td></td>
<td></td>
<td></td>
<td>1.0000000</td>
<td>0.7694165</td>
<td>4.8014378</td>
<td>24.89860</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>reno</td>
<td></td>
<td></td>
<td>0.9999741</td>
<td>0.9113953</td>
<td>20.24048</td>
<td>58.54340</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td></td>
<td></td>
<td></td>
<td>0.9999915</td>
<td>0.9698033</td>
<td>6.9873276</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>reno</td>
<td></td>
<td></td>
<td>1.0000000</td>
<td>0.9792587</td>
<td>24.68197</td>
<td>76.56740</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l4s-cred</td>
<td>1</td>
<td>1</td>
<td>cubic</td>
<td>dctcp</td>
<td></td>
<td></td>
<td>0.9864210</td>
<td>1.0003589</td>
<td>1.3134242</td>
<td>4.4714500</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l4s-cred</td>
<td>1</td>
<td>1</td>
<td>cubic</td>
<td>relentless</td>
<td></td>
<td></td>
<td></td>
<td>0.9842381</td>
<td>0.8606751</td>
<td>1.2974016</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l4s-cred</td>
<td>1</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td></td>
<td></td>
<td>0.9913302</td>
<td>1.2362902</td>
<td>1.2895002</td>
<td>4.4834600</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l4s-cred</td>
<td>1</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td></td>
<td></td>
<td></td>
<td>0.9989653</td>
<td>0.9474831</td>
<td>1.4449520</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l4s-cred</td>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td></td>
<td></td>
<td>1.0000000</td>
<td>1.0007519</td>
<td>1.3829247</td>
<td>4.4714500</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l4s-cred</td>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9864642667</td>
<td>1.1013705838</td>
<td>1.3002934871</td>
<td>4.4714500000</td>
<td>13.5116903295</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9999790600</td>
<td>1.0018207589</td>
<td>1.3697430972</td>
<td>4.4714500000</td>
<td>52.1054068975</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9524659833</td>
<td>1.2044115880</td>
<td>1.0879712926</td>
<td>4.7726700000</td>
<td>13.3551452754</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>5</td>
<td>1</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9930698333</td>
<td>0.7762237536</td>
<td>1.7821160913</td>
<td>9.4166600000</td>
<td>18.0294183055</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>5</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9869617500</td>
<td>0.8013222239</td>
<td>1.7544390103</td>
<td>10.4794000000</td>
<td>17.8475330238</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>5</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9760873167</td>
<td>0.8551807503</td>
<td>1.7110156868</td>
<td>9.5844300000</td>
<td>18.0284081664</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9944067200</td>
<td>1.2307945144</td>
<td>1.8179136631</td>
<td>9.2778400000</td>
<td>36.2873753785</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9930973667</td>
<td>1.0024256346</td>
<td>1.6605637050</td>
<td>9.2658300000</td>
<td>22.3616265907</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>5</td>
<td>5</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9990453667</td>
<td>1.0047374465</td>
<td>1.6567830953</td>
<td>8.3841800000</td>
<td>49.4996706761</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>5</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9784330333</td>
<td>1.0099676779</td>
<td>1.5705381663</td>
<td>8.3871600000</td>
<td>22.4703005885</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9952628567</td>
<td>1.5135592354</td>
<td>1.8822361027</td>
<td>9.5887600000</td>
<td>33.9164841016</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9999866833</td>
<td>0.9125598344</td>
<td>1.5447392973</td>
<td>7.4009000000</td>
<td>55.0301696059</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>5</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9921161167</td>
<td>1.4729472010</td>
<td>1.8075235770</td>
<td>9.2413000000</td>
<td>36.0031549706</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9668851567</td>
<td>0.8896215281</td>
<td>2.2373166739</td>
<td>15.6088000000</td>
<td>26.8320751213</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9725576633</td>
<td>0.8628009171</td>
<td>2.2031367099</td>
<td>12.8826000000</td>
<td>27.9050317179</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9855001333</td>
<td>1.0018575954</td>
<td>2.0219796301</td>
<td>12.8826000000</td>
<td>32.2450266446</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9998788833</td>
<td>0.8640728346</td>
<td>1.9094060970</td>
<td>11.6810000000</td>
<td>44.8084814734</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9971754167</td>
<td>0.8894077338</td>
<td>1.9331925409</td>
<td>12.0040000000</td>
<td>44.2883735085</td>
</tr>
<tr>
<td>l4s-cred</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9892573067</td>
<td>1.3258827451</td>
<td>2.4569493956</td>
<td>14.0722000000</td>
<td>57.4990952359</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9836898967</td>
<td>0.9981106340</td>
<td>2.0190110900</td>
<td>23.7090000000</td>
<td>15.7229641333</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9771563100</td>
<td>0.9631236342</td>
<td>2.3453270847</td>
<td>21.3059000000</td>
<td>15.8488953293</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9427152100</td>
<td>0.9627095364</td>
<td>2.7918459485</td>
<td>28.5154000000</td>
<td>16.3057579605</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>5</td>
<td>1</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9797997833</td>
<td>0.6569111254</td>
<td>2.2576168123</td>
<td>15.6088000000</td>
<td>15.6421680283</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>5</td>
<td>1</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9718967167</td>
<td>0.9065845477</td>
<td>2.3307557066</td>
<td>14.4072000000</td>
<td>17.6660200570</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9904388000</td>
<td>0.8284892374</td>
<td>1.7706943159</td>
<td>16.4874000000</td>
<td>15.3623127004</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9881757500</td>
<td>0.8167420388</td>
<td>1.9115541209</td>
<td>15.6208000000</td>
<td>15.4695511866</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>5</td>
<td>5</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9500650000</td>
<td>0.9607554594</td>
<td>2.8207302033</td>
<td>23.6970000000</td>
<td>16.5668226656</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9889367167</td>
<td>0.9983221079</td>
<td>1.9487440075</td>
<td>19.0267000000</td>
<td>15.6930415510</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9924467667</td>
<td>0.8834100692</td>
<td>1.7145201913</td>
<td>20.1043000000</td>
<td>15.6266945008</td>
</tr>
<tr>
<td>L4s-pi2</td>
<td>5</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9684695833</td>
<td>0.9822025756</td>
<td>2.5699124968</td>
<td>28.8384000000</td>
<td>16.1383648626</td>
</tr>
<tr>
<td>l4s-pi2</td>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9662867000</td>
<td>0.4095875227</td>
<td>2.8973571784</td>
<td>21.2938000000</td>
<td>15.8335474004</td>
</tr>
<tr>
<td>l4s-pi2</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9873546667</td>
<td>0.9766070859</td>
<td>2.2308245620</td>
<td>17.7011000000</td>
<td>16.0368345223</td>
</tr>
<tr>
<td>l4s-pi2</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9888566833</td>
<td>0.7357658858</td>
<td>2.0839103435</td>
<td>17.9687000000</td>
<td>15.7371094601</td>
</tr>
<tr>
<td>l4s-pi2</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9769609500</td>
<td>0.9586344802</td>
<td>2.4994877528</td>
<td>22.5075000000</td>
<td>16.3692229924</td>
</tr>
<tr>
<td>l4s-pi2</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9734360833</td>
<td>0.7316276451</td>
<td>2.3483889166</td>
<td>22.9834000000</td>
<td>16.0251181188</td>
</tr>
<tr>
<td>l4s-pi2</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9932553333</td>
<td>0.9985057160</td>
<td>1.8607202485</td>
<td>18.8906000000</td>
<td>15.6590192654</td>
</tr>
<tr>
<td>l4s-pi2</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9903662500</td>
<td>0.8395815488</td>
<td>1.9753231147</td>
<td>21.2938000000</td>
<td>15.6652040570</td>
</tr>
<tr>
<td>l4s-pi2</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9755094333</td>
<td>0.8285956560</td>
<td>2.2244442773</td>
<td>26.1002000000</td>
<td>15.8736843153</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>1</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9246988500</td>
<td>1.0367228246</td>
<td>1.2774631122</td>
<td>4.4714500000</td>
<td>23.5476838283</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>1</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9129832500</td>
<td>1.0811318469</td>
<td>1.2920543867</td>
<td>4.7662000000</td>
<td>30.8811668512</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9854298167</td>
<td>1.0005343628</td>
<td>1.3083522977</td>
<td>4.4714500000</td>
<td>21.0806758420</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9879355100</td>
<td>0.9496815032</td>
<td>1.4385801326</td>
<td>4.4714500000</td>
<td>18.3901121793</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>5</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9379089600</td>
<td>0.9931031674</td>
<td>1.1741170705</td>
<td>4.4714500000</td>
<td>21.2769059034</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9563333133</td>
<td>0.9437240598</td>
<td>1.3286080868</td>
<td>4.7606500000</td>
<td>19.3230289393</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9965870200</td>
<td>0.9992766062</td>
<td>1.2363516675</td>
<td>4.4714500000</td>
<td>17.3329242040</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9925816100</td>
<td>0.9986712500</td>
<td>1.2497958810</td>
<td>4.4714500000</td>
<td>17.8382160749</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9741771233</td>
<td>0.9953658301</td>
<td>1.1554221282</td>
<td>4.4714500000</td>
<td>17.8294927463</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>1</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9081243833</td>
<td>0.9523914342</td>
<td>1.6208151930</td>
<td>10.4794000000</td>
<td>28.9695798823</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>1</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9748380667</td>
<td>0.7900670786</td>
<td>1.8538424537</td>
<td>9.2778400000</td>
<td>33.9485278788</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>1</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.8786451167</td>
<td>0.9735954011</td>
<td>1.5332478466</td>
<td>8.3991800000</td>
<td>31.5160782319</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9484829333</td>
<td>0.8818538841</td>
<td>1.7610002499</td>
<td>10.4674000000</td>
<td>33.0313622739</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9492716500</td>
<td>0.8093385265</td>
<td>1.4399141902</td>
<td>7.1855600000</td>
<td>21.5625931984</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9916804500</td>
<td>0.8774926283</td>
<td>1.4221273615</td>
<td>6.8626300000</td>
<td>17.4795067335</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9789741833</td>
<td>0.8693455011</td>
<td>1.3053575930</td>
<td>7.0950900000</td>
<td>17.9757126215</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9263535167</td>
<td>0.8126948580</td>
<td>1.9035517737</td>
<td>12.8826000000</td>
<td>27.6250044633</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9325016967</td>
<td>0.9269057228</td>
<td>2.0318125756</td>
<td>11.6810000000</td>
<td>33.0430549505</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9688651500</td>
<td>0.7070949392</td>
<td>2.1176334340</td>
<td>12.8706000000</td>
<td>35.9482350974</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9174415500</td>
<td>0.9085057639</td>
<td>1.9669476174</td>
<td>12.0040000000</td>
<td>35.1100223084</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9473489000</td>
<td>0.7534119121</td>
<td>1.9315563610</td>
<td>14.0842000000</td>
<td>32.0454932194</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9766962000</td>
<td>0.9456953064</td>
<td>1.5869804916</td>
<td>9.2898600000</td>
<td>20.9602398944</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9425149500</td>
<td>0.9304348314</td>
<td>1.5373177590</td>
<td>10.7904000000</td>
<td>25.1830522529</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9756466833</td>
<td>0.7480184965</td>
<td>1.6143916378</td>
<td>10.8024000000</td>
<td>22.4629196806</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9606693000</td>
<td>0.9385200415</td>
<td>1.4947048749</td>
<td>9.6007800000</td>
<td>21.1769152896</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9425150167</td>
<td>0.7581845111</td>
<td>1.5659718460</td>
<td>14.1559000000</td>
<td>24.4479696495</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9818874333</td>
<td>0.9830240696</td>
<td>1.4841452196</td>
<td>9.6007800000</td>
<td>16.2603867072</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9935984000</td>
<td>0.8501000926</td>
<td>1.5235660899</td>
<td>9.2778400000</td>
<td>18.3412254027</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9897030167</td>
<td>0.9705983743</td>
<td>1.3794777362</td>
<td>9.2778400000</td>
<td>16.1585691529</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9507593737</td>
<td>0.5533202914</td>
<td>1.4098693644</td>
<td>11.2902000000</td>
<td>19.8910894973</td>
</tr>
</tbody>
</table>
### B.2 Data from LTE Link Simulations

<p>| sq-pie | 1 | 1 | cubic | cubic | 1 | 0.6608886195 | 0.9960126585 | 8.526904392 | 46.0220000000 | 8.526904392 | 46.0220000000 |
| sq-pie | 1 | 1 | reno | reno | 1 | 0.7410520346 | 1.0039266032 | 7.9071725089 | 25.0460000000 | 7.9071725089 | 25.0460000000 |
| sq-pie | 1 | 10 | cubic | cubic | 0 | 0.9819141738 | 1.0012682321 | 5.2639553014 | 18.0580000000 | 5.2639553014 | 18.0580000000 |
| sq-pie | 1 | 10 | reno | reno | 0 | 0.8295020512 | 1.000954370 | 6.182696439 | 23.0460000000 | 6.182696439 | 23.0460000000 |
| sq-pie | 5 | 1 | cubic | cubic | 1 | 0.7815967960 | 1.0004151478 | 7.0957104028 | 35.0460000000 | 7.0957104028 | 35.0460000000 |
| sq-pie | 5 | 1 | reno | reno | 1 | 0.8289245940 | 1.0108949233 | 6.4518648075 | 27.0340000000 | 6.4518648075 | 27.0340000000 |
| sq-pie | 5 | 10 | cubic | cubic | 0 | 0.9829951511 | 0.9990055621 | 5.2540700653 | 22.0340000000 | 5.2540700653 | 22.0340000000 |
| sq-pie | 5 | 10 | reno | reno | 1 | 0.9577050666 | 0.9999811733 | 5.9559626682 | 21.0460000000 | 5.9559626682 | 21.0460000000 |
| sq-pie | 5 | 5 | cubic | cubic | 0 | 0.3258040145 | 0.002619337 | 6.6595530142 | 30.0460000000 | 6.6595530142 | 30.0460000000 |
| sq-pie | 5 | 5 | reno | reno | 1 | 0.308740825 | 1.00000028370 | 6.1512559734 | 30.0460000000 | 6.1512559734 | 30.0460000000 |
| sq-pie | 10 | 10 | cubic | cubic | 0 | 0.9386838658 | 1.0005769302 | 5.2033008918 | 27.0460000000 | 5.2033008918 | 27.0460000000 |
| sq-pie | 10 | 10 | reno | reno | 1 | 0.9129797965 | 1.0001037917 | 5.4540158143 | 24.0460000000 | 5.4540158143 | 24.0460000000 |
| sq-red | 1 | 1 | cubic | cubic | 1 | 0.4216681784 | 0.9753971766 | 6.9465955971 | 11.0460000000 | 6.9465955971 | 11.0460000000 |
| sq-red | 1 | 5 | cubic | cubic | 0 | 0.6689662674 | 1.0545391082 | 6.6605659581 | 11.0460000000 | 6.6605659581 | 11.0460000000 |
| sq-red | 1 | 5 | reno | reno | 1 | 0.3647736357 | 0.975760205 | 5.4238635045 | 11.0460000000 | 5.4238635045 | 11.0460000000 |
| sq-red | 1 | 10 | cubic | cubic | 1 | 0.3891286036 | 0.997265922 | 7.5129339478 | 14.0460000000 | 7.5129339478 | 14.0460000000 |
| sq-red | 1 | 10 | reno | reno | 1 | 0.689662674 | 1.0545391082 | 6.650313214 | 13.0460000000 | 6.650313214 | 13.0460000000 |
| sq-red | 5 | 5 | cubic | cubic | 1 | 0.9760161603 | 0.0008398965 | 6.339079163 | 19.0460000000 | 6.339079163 | 19.0460000000 |
| sq-red | 5 | 5 | reno | reno | 1 | 0.9932163223 | 0.010491616 | 6.3645372907 | 14.0460000000 | 6.3645372907 | 14.0460000000 |
| sq-red | 10 | 10 | cubic | cubic | 1 | 0.860240202 | 0.017445238 | 5.8687690124 | 23.0460000000 | 5.8687690124 | 23.0460000000 |
| sq-red | 10 | 10 | reno | reno | 1 | 0.4733558736 | 1.3422714378 | 5.9157666262 | 14.0460000000 | 5.9157666262 | 14.0460000000 |
| eq-red | 10 | 10 | cubic | cubic | 1 | 0.3580116036 | 0.0134683096 | 8.347622335 | 24.0460000000 | 8.347622335 | 24.0460000000 |
| sq-red | 1  | 10 | reno | reno | 0 | 0.6498112930 | 1.2123819524 | 6.4217952590 | 13.0580000000 | 6.4217952590 | 13.0580000000 |
| sq-red | 5  | 1  | cubic | cubic | 0 | 0.9722545541 | 0.9996181511 | 6.2443593084 | 18.0340000000 | 6.2443593084 | 18.0340000000 |
| sq-red | 5  | 1  | reno | reno | 0 | 0.8704333904 | 0.9990155680 | 6.4142959470 | 20.0580000000 | 6.4142959470 | 20.0580000000 |
| sq-red | 5  | 1  | reno | reno | 1 | 0.8653356399 | 0.9994357688 | 6.5368259874 | 21.0580000000 | 6.5368259874 | 21.0580000000 |
| sq-red | 5  | 5  | cubic | cubic | 1 | 0.9742533086 | 0.9998477300 | 8.3279297639 | 25.0440000000 | 8.3279297639 | 25.0440000000 |
| sq-red | 5  | 5  | reno | reno | 1 | 0.8407934683 | 1.0002681848 | 8.0455937475 | 27.0580000000 | 8.0455937475 | 27.0580000000 |
| sq-red | 5  | 10 | reno | reno | 0 | 0.9041831432 | 1.4385297229 | 6.6784009146 | 21.0220000000 | 6.6784009146 | 21.0220000000 |
| sq-red | 10 | 1  | cubic | cubic | 0 | 0.9750654573 | 1.0021470327 | 8.5989347543 | 22.0460000000 | 8.5989347543 | 22.0460000000 |
| sq-red | 10 | 1  | cubic | cubic | 1 | 0.9721687697 | 0.9989637917 | 8.9338419297 | 23.0460000000 | 8.9338419297 | 23.0460000000 |
| sq-red | 10 | 1  | reno | reno | 0 | 0.8838226659 | 1.0010734792 | 8.1194077106 | 24.0580000000 | 8.1194077106 | 24.0580000000 |
| sq-red | 10 | 5  | cubic | cubic | 0 | 0.9819563225 | 1.1195153203 | 9.0810224835 | 22.0100000000 | 9.0810224835 | 22.0100000000 |
| dq-pie | 1  | 1  | reno | dctcp | 1 | 0.9990122837 | 1.0000000000 | 5.4458590000 | 8.0460000000 | 297.4939205330 | 393.0460000000 |
| dq-pie | 1  | 1  | reno | relentless | 0 | 0.9987922894 | 0.9999998972 | 5.8284749975 | 18.0460000000 | 341.7315031410 | 429.0460000000 |
| dq-pie | 1  | 1  | reno | relentless | 1 | 0.9988043830 | 0.9999999243 | 5.5857490666 | 16.0460000000 | 289.3586272250 | 387.0460000000 |
| dq-pie | 1  | 5  | cubic | cubic | 0 | 0.9819553812 | 1.3556391663 | 5.0124378737 | 46.0460000000 | 388.3436356180 | 684.0460000000 |
| dq-pie | 1  | 5  | reno | dctcp | 0 | 0.9985656589 | 1.4439891413 | 4.7754463911 | 12.0460000000 | 333.9261671440 | 528.0460000000 |
| dq-pie | 1  | 5  | reno | dctcp | 1 | 0.9972039836 | 1.4411214848 | 5.2029217092 | 11.0460000000 | 337.3068435040 | 532.0460000000 |
| dq-pie | 1  | 10 | cubic | cubic | 0 | 0.9841784845 | 1.6215090010 | 5.4197526019 | 46.0460000000 | 377.8416619170 | 713.0340000000 |
| dq-pie | 1 | 10 | cubic | dctcp | 0 | 0.9970632915 | 1.6678909226 | 4.9751905485 | 12.0460000000 | 412.1928898530 | 712.0340000000 |
| dq-pie | 1 | 10 | cubic | relentless | 0 | 0.9983815744 | 1.6693531404 | 5.7051486500 | 16.0460000000 | 419.0540756450 | 710.0460000000 |
| dq-pie | 1 | 10 | reno | dctcp | 0 | 0.9990122932 | 1.6694214927 | 5.6812091833 | 8.0460000000 | 358.4828760170 | 538.0460000000 |
| dq-pie | 1 | 10 | reno | relentless | 0 | 0.9986193002 | 1.6692085144 | 5.8099481500 | 16.0460000000 | 358.8380455670 | 540.0460000000 |
| dq-pie | 5 | 1 | cubic | cubic | 0 | 0.9914926317 | 0.5359202425 | 6.0038052829 | 23.0460000000 | 361.7345001820 | 621.0560000000 |
| dq-pie | 5 | 1 | cubic | cubic | 1 | 0.9927733314 | 0.5348300422 | 5.8685597817 | 28.0440000000 | 240.9375563450 | 737.0460000000 |
| dq-pie | 5 | 1 | cubic | relentless | 1 | 0.9953293687 | 0.5487527961 | 5.0890080601 | 21.0440000000 | 223.3307355350 | 735.0460000000 |
| dq-pie | 5 | 1 | reno | dctcp | 0 | 0.9964733980 | 0.5515218246 | 5.1030353549 | 16.0460000000 | 338.7133731350 | 429.0460000000 |
| dq-pie | 5 | 1 | reno | relentless | 0 | 0.9952690911 | 0.5484744445 | 5.0921501928 | 20.0440000000 | 337.8886928100 | 429.0460000000 |
| dq-pie | 5 | 5 | cubic | relentless | 1 | 0.9951473569 | 0.9998968674 | 5.0966119328 | 19.0460000000 | 322.6445783090 | 944.0340000000 |
| dq-pie | 5 | 5 | reno | dctcp | 0 | 0.9967019205 | 0.9999653672 | 5.0627607219 | 14.0460000000 | 372.6407107640 | 529.0460000000 |
| dq-pie | 5 | 5 | reno | relentless | 0 | 0.9952122076 | 0.9998742627 | 5.0676541535 | 17.0340000000 | 367.6782262820 | 530.0460000000 |
| dq-pie | 5 | 5 | reno | relentless | 1 | 0.9950313368 | 0.9998666455 | 5.1410954444 | 19.0460000000 | 359.9858585710 | 532.0460000000 |
| dq-pie | 5 | 5 | reno | reno | 0 | 0.9868194809 | 0.9935013662 | 6.2068872608 | 34.0440000000 | 391.1377608230 | 542.0460000000 |
| dq-pie | 5 | 10 | cubic | dctcp | 1 | 0.9964542974 | 1.1075632986 | 5.0185955255 | 14.0460000000 | 412.3959122660 | 1084.0580000000 |
| dq-pie | 5 | 10 | cubic | relentless | 0 | 0.9948027001 | 1.1051603047 | 5.0831598369 | 17.0460000000 | 408.0786554340 | 711.0460000000 |
| dq-pie | 5 | 10 | reno | relentless | 1 | 0.9950749857 | 1.1049651293 | 5.1514445152 | 20.0580000000 | 346.6130671020 | 556.0460000000 |
| dq-pie | 10 | 1 | cubic | dctcp | 0 | 0.9933362331 | 0.3234418557 | 5.0036845055 | 18.0220000000 | 368.6090872220 | 602.0460000000 |
| dq-pie | 10 | 1 | cubic | relentless | 0 | 0.9897250333 | 0.3149634353 | 5.2277955636 | 34.0340000000 | 355.2105538230 | 654.0460000000 |
| dq-pie | 10 | 1 | cubic | relentless | 1 | 0.9913461590 | 0.3153342334 | 5.1123041990 | 25.0340000000 | 216.6893717230 | 747.0460000000 |</p>
<table>
<thead>
<tr>
<th>dq</th>
<th>10</th>
<th>1</th>
<th>reno</th>
<th>relentless</th>
<th>1</th>
<th>0.9915314984</th>
<th>0.3161628080</th>
<th>5.1513647071</th>
<th>21.0460000000</th>
<th>284.7417995740</th>
<th>388.0460000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>dq</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>reno</td>
<td>0</td>
<td>0.9861413482</td>
<td>0.2875826597</td>
<td>5.5995525743</td>
<td>32.0440000000</td>
<td>322.1547441800</td>
<td>444.0460000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>cubic</td>
<td>1</td>
<td>0.9907546587</td>
<td>0.8613776255</td>
<td>5.2793016969</td>
<td>27.0440000000</td>
<td>328.8940834690</td>
<td>969.0340000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9939364708</td>
<td>0.8789963345</td>
<td>5.0471855161</td>
<td>18.0460000000</td>
<td>322.0624584980</td>
<td>950.0580000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>5</td>
<td>relentless</td>
<td>relentless</td>
<td>1</td>
<td>0.9914771535</td>
<td>0.8707413320</td>
<td>5.0961558532</td>
<td>20.0440000000</td>
<td>326.7042642270</td>
<td>917.0460000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9939092793</td>
<td>0.8780390434</td>
<td>5.0414576875</td>
<td>18.0460000000</td>
<td>340.8705228570</td>
<td>530.0460000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9941723712</td>
<td>0.8794489356</td>
<td>5.0583772138</td>
<td>16.0460000000</td>
<td>344.9988913990</td>
<td>540.0460000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9913299487</td>
<td>0.8699260494</td>
<td>5.1660218545</td>
<td>24.0460000000</td>
<td>337.4930284930</td>
<td>534.0460000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>cubic</td>
<td>0</td>
<td>0.9902363662</td>
<td>0.9980707492</td>
<td>5.2590624319</td>
<td>32.0440000000</td>
<td>417.5256009510</td>
<td>723.0580000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9936848926</td>
<td>0.9997251064</td>
<td>5.0722805193</td>
<td>17.0460000000</td>
<td>400.9912227430</td>
<td>716.0460000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9908477657</td>
<td>0.9990981724</td>
<td>5.1354336363</td>
<td>21.0340000000</td>
<td>407.5668916330</td>
<td>712.0340000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9945716106</td>
<td>0.9998078698</td>
<td>5.0535609617</td>
<td>18.0340000000</td>
<td>353.8619832950</td>
<td>543.0580000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9941138334</td>
<td>0.9997586313</td>
<td>5.0179517449</td>
<td>17.0460000000</td>
<td>357.3190413060</td>
<td>558.0460000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9913299487</td>
<td>0.8699260494</td>
<td>5.1660218545</td>
<td>24.0460000000</td>
<td>337.4930284930</td>
<td>534.0460000000</td>
</tr>
<tr>
<td>dq</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9912992774</td>
<td>0.8275909061</td>
<td>3.2403689588</td>
<td>19.0460000000</td>
<td>326.7042642270</td>
<td>917.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>1</td>
<td>5</td>
<td>reno</td>
<td>reno</td>
<td>0</td>
<td>0.9769135197</td>
<td>1.1510757926</td>
<td>1.6536501207</td>
<td>9.0460000000</td>
<td>211.0204022080</td>
<td>354.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>cubic</td>
<td>1</td>
<td>0.9918192337</td>
<td>0.9975437177</td>
<td>1.2651158404</td>
<td>6.0440000000</td>
<td>192.1934298780</td>
<td>265.0580000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9676072637</td>
<td>1.5679450358</td>
<td>1.3563705623</td>
<td>6.0460000000</td>
<td>356.0922144360</td>
<td>506.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9688397034</td>
<td>1.5344198131</td>
<td>1.3569847937</td>
<td>6.0460000000</td>
<td>339.4811077670</td>
<td>494.0340000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9698234740</td>
<td>1.4515360547</td>
<td>1.2648070298</td>
<td>6.0460000000</td>
<td>302.1671854700</td>
<td>418.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9705736737</td>
<td>1.4377124635</td>
<td>1.2660535894</td>
<td>6.0100000000</td>
<td>248.2729476940</td>
<td>315.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>reno</td>
<td>1</td>
<td>0.9922150295</td>
<td>0.9853768780</td>
<td>1.3099224104</td>
<td>5.0460000000</td>
<td>162.0350224510</td>
<td>208.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>1</td>
<td>cubic</td>
<td>cubic</td>
<td>1</td>
<td>0.9977025100</td>
<td>0.5504656057</td>
<td>14.4097183682</td>
<td>36.0460000000</td>
<td>218.2432762910</td>
<td>348.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>1</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9929422324</td>
<td>0.5219658105</td>
<td>16.7925658158</td>
<td>43.0440000000</td>
<td>227.4479327110</td>
<td>323.0580000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9919357768</td>
<td>0.5194746578</td>
<td>16.2752250330</td>
<td>42.0460000000</td>
<td>179.8392044540</td>
<td>293.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>cubic</td>
<td>1</td>
<td>0.9919553908</td>
<td>0.9974221691</td>
<td>7.6787256231</td>
<td>27.0220000000</td>
<td>318.4593129850</td>
<td>465.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9989940388</td>
<td>0.9999999973</td>
<td>15.8240806527</td>
<td>31.0340000000</td>
<td>332.4385426640</td>
<td>471.0340000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9857183495</td>
<td>0.9879494739</td>
<td>8.2435415977</td>
<td>31.0460000000</td>
<td>306.1784321770</td>
<td>458.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9863371839</td>
<td>0.9889717488</td>
<td>8.2280211328</td>
<td>30.0100000000</td>
<td>257.2966415050</td>
<td>360.0220000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>5</td>
<td>reno</td>
<td>reno</td>
<td>1</td>
<td>0.9853697661</td>
<td>0.9792116363</td>
<td>6.9339959454</td>
<td>29.0460000000</td>
<td>244.0820784320</td>
<td>349.0100000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9980903309</td>
<td>1.1097636321</td>
<td>9.2847768821</td>
<td>24.0220000000</td>
<td>413.1169738170</td>
<td>645.0340000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9833481080</td>
<td>1.0421378276</td>
<td>5.5961640289</td>
<td>24.0220000000</td>
<td>369.1600584030</td>
<td>543.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9980483267</td>
<td>1.1095808026</td>
<td>9.2137979801</td>
<td>25.0340000000</td>
<td>308.1574917820</td>
<td>382.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>5</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9835729036</td>
<td>1.0441620188</td>
<td>5.5050920659</td>
<td>24.0460000000</td>
<td>284.5939789800</td>
<td>393.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>cubic</td>
<td>1</td>
<td>0.9957934303</td>
<td>0.3174294968</td>
<td>18.8021078877</td>
<td>46.0340000000</td>
<td>227.0854311370</td>
<td>391.0460000000</td>
</tr>
<tr>
<td>Protocol</td>
<td>Rate</td>
<td>Mode</td>
<td>IP Version</td>
<td>L4S Version</td>
<td>0</td>
<td>0.9990122932</td>
<td>0.3305785073</td>
<td>29.6036022833</td>
<td>52.0340000000</td>
<td>220.6084715500</td>
<td>349.0460000000</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
<td>-------------</td>
<td>---</td>
<td>---------------</td>
<td>---------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9990122932</td>
<td>0.3305785073</td>
<td>29.6036022833</td>
<td>52.0340000000</td>
<td>220.6084715500</td>
<td>349.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9990122932</td>
<td>0.3305785073</td>
<td>29.6851666167</td>
<td>53.0460000000</td>
<td>257.3728690670</td>
<td>321.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9990122932</td>
<td>0.3305785073</td>
<td>29.4416891833</td>
<td>52.0340000000</td>
<td>182.4321095330</td>
<td>288.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9990122932</td>
<td>0.3305785073</td>
<td>29.7838370500</td>
<td>54.0580000000</td>
<td>181.8216517330</td>
<td>289.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>reno</td>
<td>1</td>
<td>0.9990122932</td>
<td>0.3305785073</td>
<td>29.7838370500</td>
<td>54.0580000000</td>
<td>181.8216517330</td>
<td>289.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>cubic</td>
<td>1</td>
<td>0.9917924225</td>
<td>0.8478655294</td>
<td>11.5533272808</td>
<td>40.0340000000</td>
<td>306.3607823070</td>
<td>411.0220000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9847785891</td>
<td>0.8099046610</td>
<td>12.0341807778</td>
<td>46.0340000000</td>
<td>290.2972513140</td>
<td>477.0340000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9990013406</td>
<td>0.8888725164</td>
<td>20.2845589260</td>
<td>44.0340000000</td>
<td>278.4616991670</td>
<td>347.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9851166381</td>
<td>0.8115454640</td>
<td>12.0250311767</td>
<td>45.0340000000</td>
<td>255.5775786560</td>
<td>372.0340000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9845728561</td>
<td>0.8077343025</td>
<td>12.0336953460</td>
<td>45.0460000000</td>
<td>245.7785679120</td>
<td>380.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>cubic</td>
<td>0</td>
<td>0.9899971953</td>
<td>0.9933927907</td>
<td>8.6635395985</td>
<td>34.0460000000</td>
<td>368.5067930460</td>
<td>529.0100000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>cubic</td>
<td>1</td>
<td>0.9903285796</td>
<td>0.9940731357</td>
<td>8.7447502857</td>
<td>31.0440000000</td>
<td>370.8389551110</td>
<td>504.0220000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9832209260</td>
<td>0.9810744210</td>
<td>9.6778347045</td>
<td>38.0340000000</td>
<td>352.7161818860</td>
<td>481.0460000000</td>
</tr>
<tr>
<td>dq-red</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9981945997</td>
<td>0.9999933712</td>
<td>15.3804128431</td>
<td>38.0460000000</td>
<td>313.3524634800</td>
<td>387.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>1</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.8355064214</td>
<td>0.6066676361</td>
<td>2.1545425109</td>
<td>11.0460000000</td>
<td>12.3028151730</td>
<td>56.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>1</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.8227625956</td>
<td>0.6389161397</td>
<td>2.6865532918</td>
<td>11.0460000000</td>
<td>11.5784864139</td>
<td>52.0340000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>1</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.7743599677</td>
<td>0.6538629779</td>
<td>2.9044144737</td>
<td>11.0460000000</td>
<td>11.0882132961</td>
<td>53.0340000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.8389654155</td>
<td>0.6076083152</td>
<td>2.2882472345</td>
<td>11.0340000000</td>
<td>11.5459795259</td>
<td>45.0340000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9744725062</td>
<td>0.9973062121</td>
<td>1.1565007836</td>
<td>5.0460000000</td>
<td>17.3636904852</td>
<td>57.0100000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>10</td>
<td>cubic</td>
<td>relentless</td>
<td>0</td>
<td>0.9915500894</td>
<td>0.9603139373</td>
<td>1.1406492587</td>
<td>5.0460000000</td>
<td>15.6192571888</td>
<td>45.9980000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>0</td>
<td>0.9677009527</td>
<td>0.9959099293</td>
<td>1.1380703223</td>
<td>8.9840000000</td>
<td>16.8960062405</td>
<td>55.9980000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9511838258</td>
<td>0.9931344664</td>
<td>1.1337641736</td>
<td>6.0080000000</td>
<td>16.9134480175</td>
<td>59.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>1</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9699539760</td>
<td>0.9603043583</td>
<td>1.1609400135</td>
<td>5.0220000000</td>
<td>16.4454036999</td>
<td>51.0340000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>1</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.8684255467</td>
<td>0.3704662245</td>
<td>2.3651058741</td>
<td>14.0580000000</td>
<td>11.5005771017</td>
<td>66.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>1</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.8274893041</td>
<td>0.2924644078</td>
<td>2.9554394678</td>
<td>14.0460000000</td>
<td>10.1335429563</td>
<td>64.0340000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.7847186062</td>
<td>0.2896471365</td>
<td>2.4860719462</td>
<td>14.0220000000</td>
<td>9.6781565896</td>
<td>46.0340000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.7912155638</td>
<td>0.2910284156</td>
<td>2.4517296986</td>
<td>15.0440000000</td>
<td>10.4176676881</td>
<td>46.0100000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9239107910</td>
<td>0.8530367005</td>
<td>1.8915958424</td>
<td>17.0340000000</td>
<td>17.6422659473</td>
<td>71.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>5</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.9649607055</td>
<td>0.7403898422</td>
<td>1.4476514653</td>
<td>12.0580000000</td>
<td>18.5657381502</td>
<td>54.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>5</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.8813861285</td>
<td>0.8348191605</td>
<td>2.1792538281</td>
<td>17.0460000000</td>
<td>19.5028006869</td>
<td>73.0100000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>5</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.8958078532</td>
<td>0.7517457729</td>
<td>1.6543737028</td>
<td>14.0580000000</td>
<td>19.2233451743</td>
<td>61.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>0</td>
<td>0.9638951322</td>
<td>0.9742451013</td>
<td>1.4153409111</td>
<td>8.0340000000</td>
<td>17.0808866407</td>
<td>47.0580000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9458730272</td>
<td>0.9582732730</td>
<td>1.4631694530</td>
<td>11.0460000000</td>
<td>17.4936083898</td>
<td>75.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>5</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9266942575</td>
<td>0.9435323302</td>
<td>1.5238799736</td>
<td>12.0440000000</td>
<td>17.8359193981</td>
<td>68.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>1</td>
<td>cubic</td>
<td>relentless</td>
<td>1</td>
<td>0.8524181118</td>
<td>0.1958404899</td>
<td>2.5381366858</td>
<td>17.0560000000</td>
<td>11.8962095458</td>
<td>62.0340000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>1</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.8259064176</td>
<td>0.1964974555</td>
<td>2.6576568428</td>
<td>18.0340000000</td>
<td>10.6758596286</td>
<td>46.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>5</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9442206978</td>
<td>0.7406554448</td>
<td>2.3604171912</td>
<td>15.0560000000</td>
<td>17.9663873143</td>
<td>61.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>10</td>
<td>cubic</td>
<td>dctcp</td>
<td>1</td>
<td>0.9526105343</td>
<td>0.9008928471</td>
<td>1.9474370619</td>
<td>11.0560000000</td>
<td>17.2648376320</td>
<td>62.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>dctcp</td>
<td>1</td>
<td>0.9132164005</td>
<td>0.8796251528</td>
<td>2.0351247095</td>
<td>14.0560000000</td>
<td>18.8979972454</td>
<td>88.0220000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>0</td>
<td>0.9449737973</td>
<td>0.7454444304</td>
<td>1.5810303367</td>
<td>12.9490000000</td>
<td>17.9215454433</td>
<td>58.0460000000</td>
</tr>
<tr>
<td>l4s-pie</td>
<td>10</td>
<td>10</td>
<td>reno</td>
<td>relentless</td>
<td>1</td>
<td>0.9299741110</td>
<td>0.7437206168</td>
<td>1.6112951049</td>
<td>11.9610000000</td>
<td>17.5746614105</td>
<td>67.0460000000</td>
</tr>
</tbody>
</table>
Appendix C

Original Problem

Evaluation of AQM schemes to support Low Latency in the Internet

Master thesis

Background

Many modern applications require low latency transmission in the Internet which is not explicitly supported today. Most networks are optimized for high throughput and low loss. This increases latency due to high queuing delays. Only very few applications actually need both high throughput and low latency, therefore the network could offer a new parallel Internet service that provides lower latency compared to the best effort service that is usually offered today. This can be achieved by separating flows into different queues at the bottleneck link that operate either independently or could be coupled as proposed by the DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput (4LS) [1].

Thesis Goals

The goal of this project is to evaluate different solutions for the realization of such an low latency service, including DualQ for 4LS.

This leads to the following tasks:

1. Implementation and configuration of different AQM setups in the ns-3 network simulator [2].
2. Design of an experimental setup focusing on traffic conditions and cases that may have scalability or fairness issues.
3. Evaluation and representation of simulation results.

Contact: Mirja Kühlewind, mirja.kuehlewind@tik.ee.ethz.ch, ETZ H93
Brian Trammell, trammell@tik.ee.ethz.ch, ETZ H93

Professor: Prof. Laurent Vanbever

References:

2. ns-3: https://www.nsnam.org/


